UNIT I: Graphics Primitives

Primitives: Basic Graphics Primitives

I. Scan conversion a Line.

II. Scan conversion Circle.

III. Scan conversion Ellipse.

- Filled area Primitives.

I. Scan conversion a Line

- It is a process of representing graphics objects a collection of pixels.
- The graphics objects are continuous. The pixels used are discrete. Each pixel can have either on or off state.
- The circuitry of the video display device of the computer is capable of converting binary values (0, 1) into a pixel on and pixel off information. 0 is represented by pixel off. 1 is represented using pixel on. Using this ability graphics computer represent picture having discrete dots.
- Any model of *graphics* can be reproduced with a dense matrix of dots or points. Most human beings think graphics objects as points, lines, circles, ellipses. For generating graphical object, many algorithms have been developed.

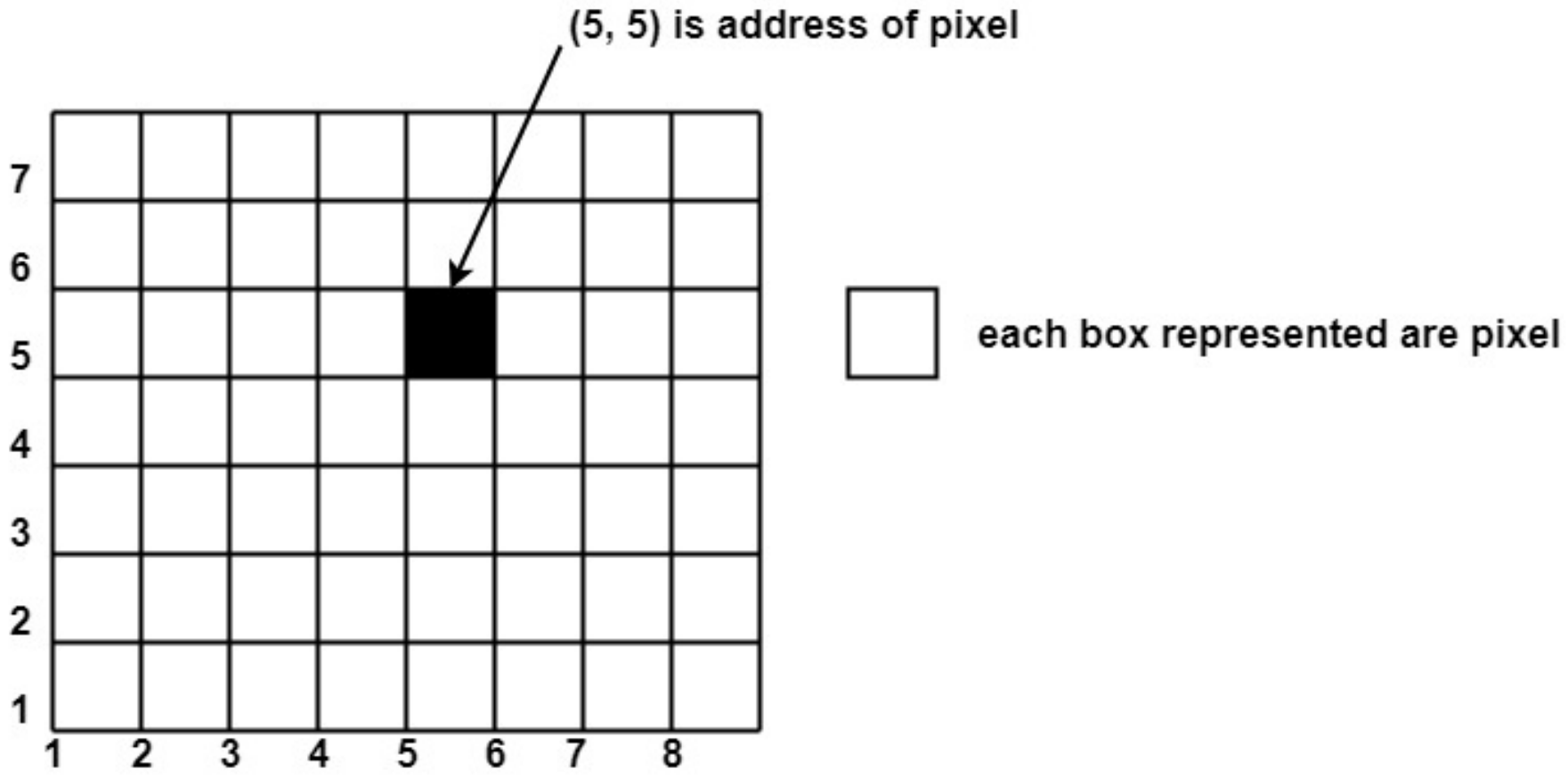
Advantages for scan conversion a line

- Algorithms can generate graphics objects at a faster rate.
- Using algorithms memory can be used efficiently.
- Algorithms can develop a higher level of graphical objects.

Pixel or Pel

- The term pixel is a short form of the picture element. It is also called a point or dot. It is the smallest picture unit accepted by display devices.
- A picture is constructed from hundreds of such pixels. Pixels are generated using commands. Lines, circle, arcs, characters; curves are drawn with closely spaced pixels. To display the digit or letter matrix of pixels is used.
- The closer the dots or pixels are, the better will be the quality of picture. Closer the dots are, crisper will be the picture.
- Picture will not appear jagged and unclear if pixels are closely spaced. So the quality of the picture is directly proportional to the density of pixels on the screen.

Pixel or Pel…


(5, 5) is address of pixel
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8
each box represented are pixel

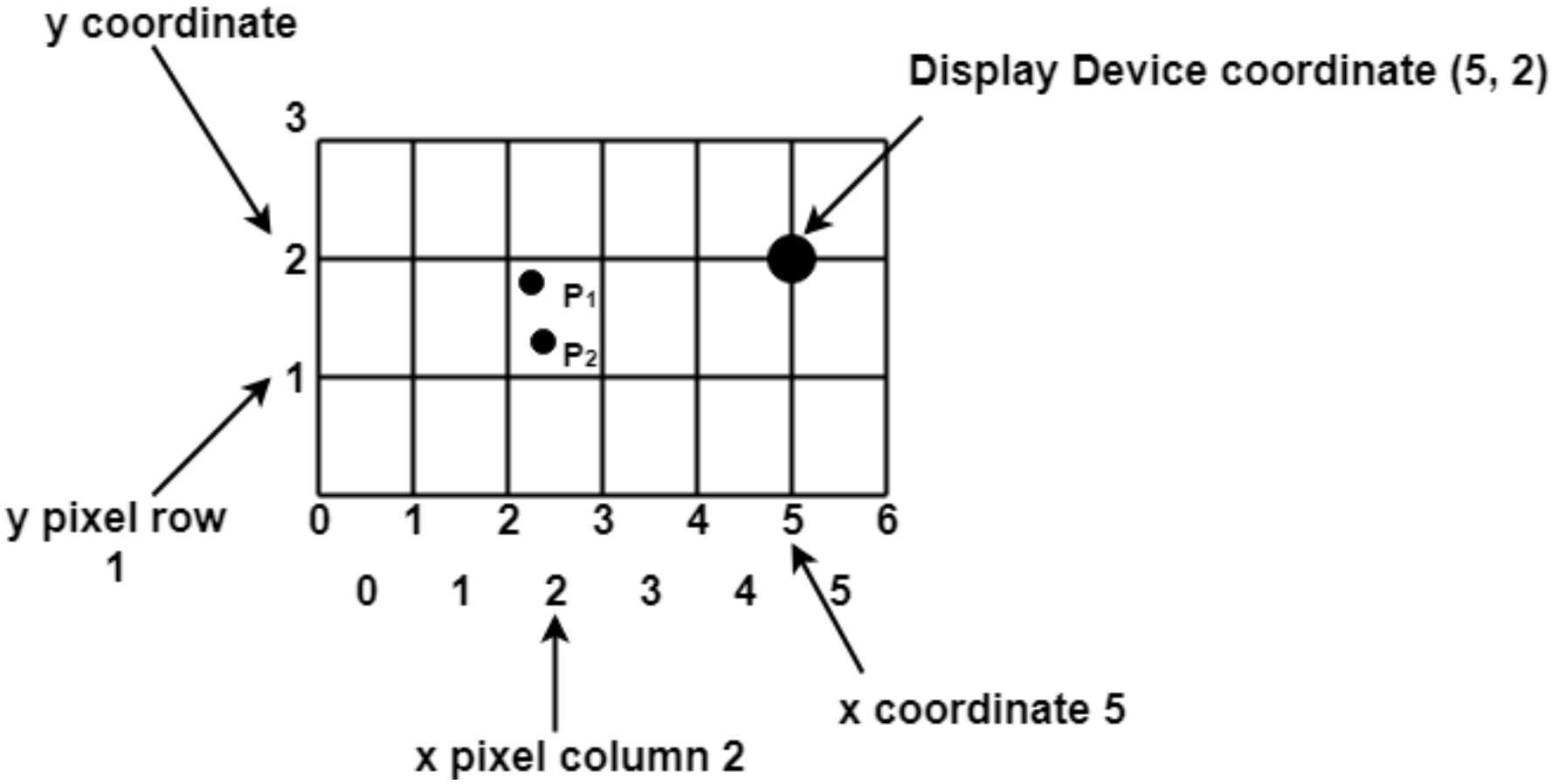
Pixel or Pel…

- Different graphics objects can be generated by setting the different intensity of pixels and different colors of pixels. Each pixel has some co-ordinate value. The coordinate is represented using row and column.
- P (5, 5) used to represent a pixel in the 5th row and the 5th column. Each pixel has some intensity value which is represented in memory of computer called a **frame buffer**.
- Frame Buffer is also called a refresh buffer. This memory is a storage area for storing pixels values using which pictures are displayed. It is also called as digital memory.
- Inside the buffer, image is stored as a pattern of binary digits either 0 or 1. So there is an array of 0 or 1 used to represent the picture. In black and white monitors, black pixels are represented using 1's and white pixels are represented using 0's. In case of systems having one bit per pixel frame buffer is called a bitmap. In systems with multiple bits per pixel it is called a pixmap.

Scan converting a Point

- Each pixel on the graphics display does not represent a mathematical point.
- Instead, it means a region which theoretically can contain an infinite number of points.
- Scan-Converting a point involves illuminating the pixel that contains the point.

Scan converting a Point…



Scan converting a Straight Line

- A straight-line segment in a scene is defined by the coordinate positions for the endpoints of the segment.
- To display the line on a raster monitor, the graphics system must first project the endpoints to integer screen coordinates and determine the nearest pixel positions along the line path between the two endpoints.
- Then the line color is loaded into the frame buffer at the corresponding pixel coordinates.

Scan converting a Straight Line…

- Reading from the frame buffer, the video controller plots the screen pixels.
- This process digitizes the line into a set of discrete integer positions that, in general, only approximates the actual line path.

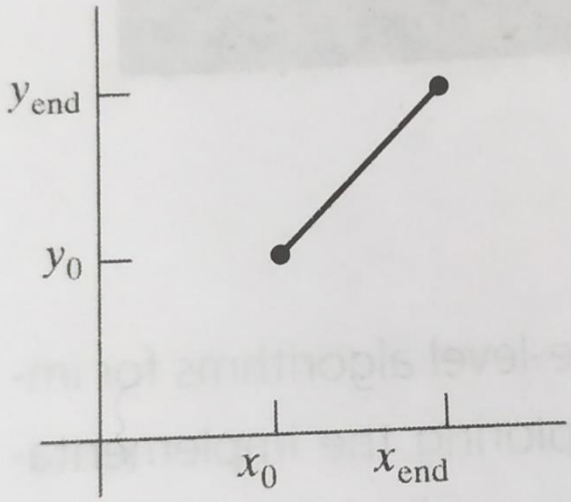


FIGURE 2
Line path between endpoint positions (x_0, y_0) and (x_{end}, y_{end}).

$$y = m \cdot x + b$$

and b as the y intercept. (
ed at positions (x_0, y_0) a
values for the slope m

$$m = \frac{y_{end} - y_0}{x_{end} - x_0}$$

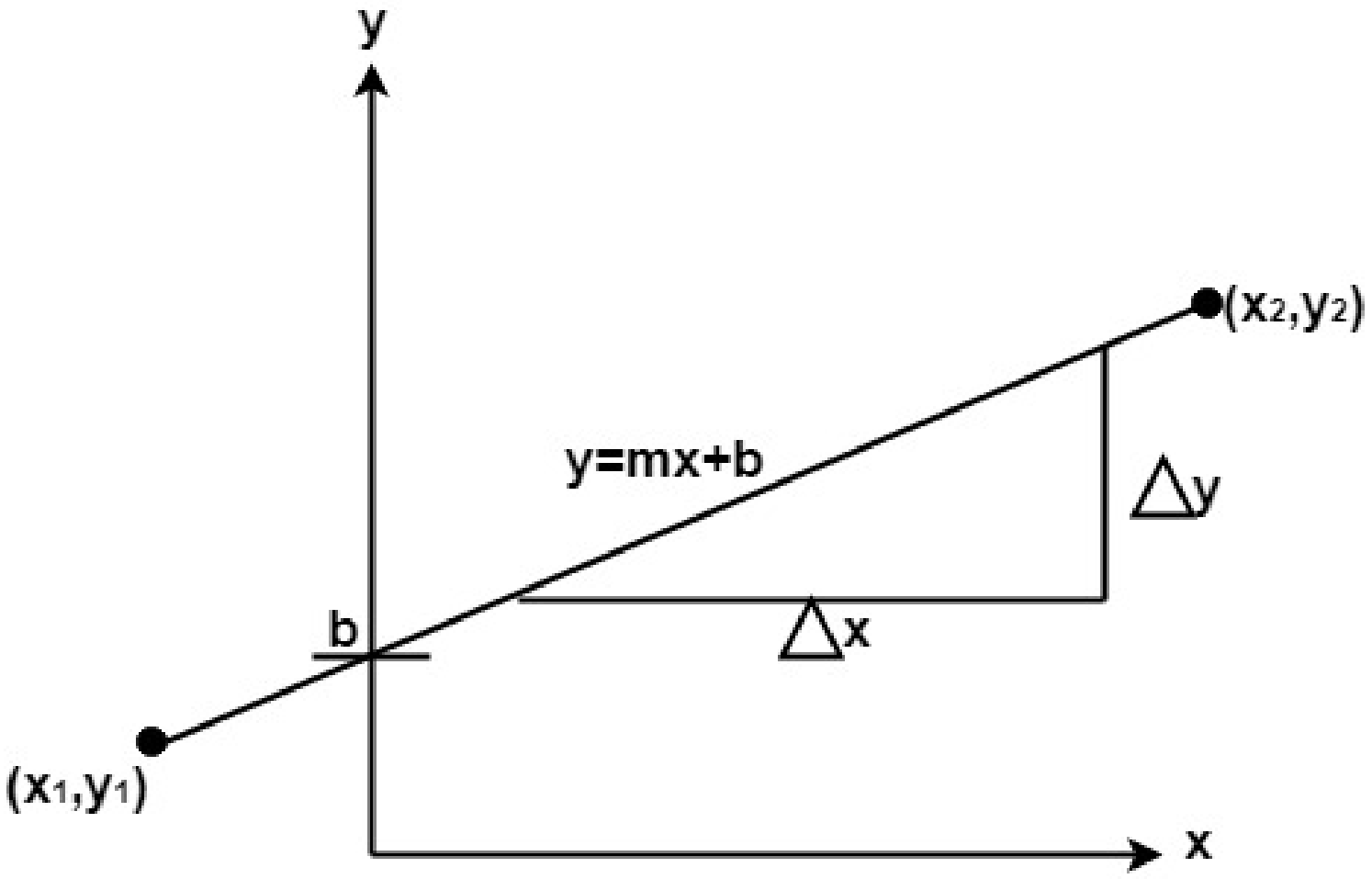
$$b = y_0 - m \cdot x_0$$

straight lines are based c
and 3.
l δx along a line, we can
n 2 as

$$\delta y = m \cdot \delta x$$

e x interval δx correspond

$$\delta x = \frac{\delta y}{m}$$


y
(x2,y2)
y=mx+b
Δy
b
Δx
(x1,y1)
x

Line Equations

- We determine pixel positions along a straight-line path from the geometric properties of the line.
- The Cartesian slope-intercept equation for a straight-line is: $y=m.x + b$

Properties of Good Line Drawing Algo.:

- Line should appear Straight.
- Lines should terminate accurately.
- Lines should have constant density.
- Line density should be independent of line length and angle.
- Line should be drawn rapidly.

Fig: O/P from a poor line generating algorithm

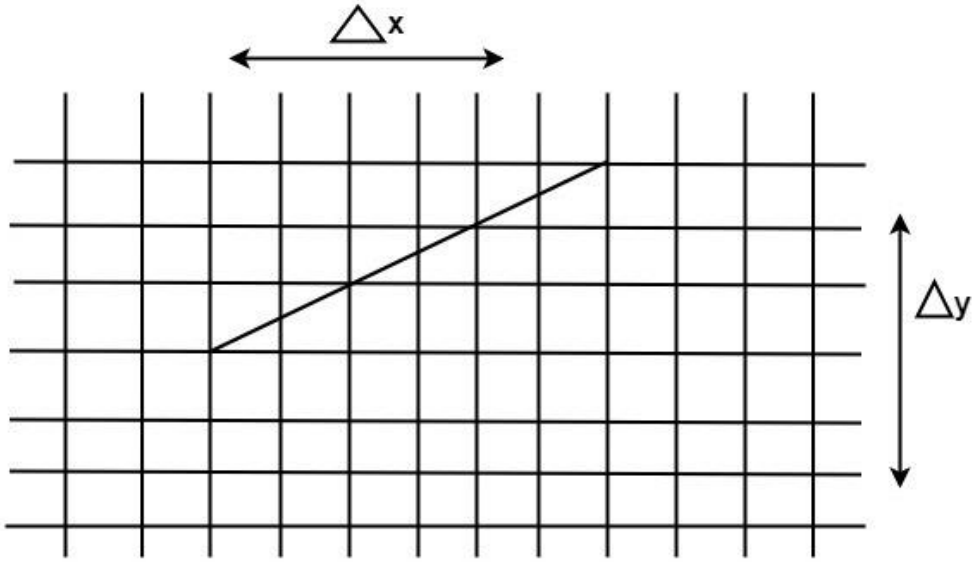


Fig: A straight line segment connecting 2 grid intersection may fail to pass through any other grid intersections.

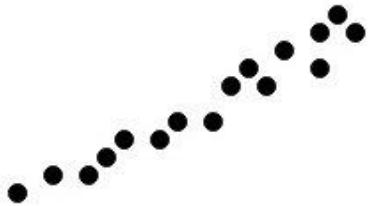
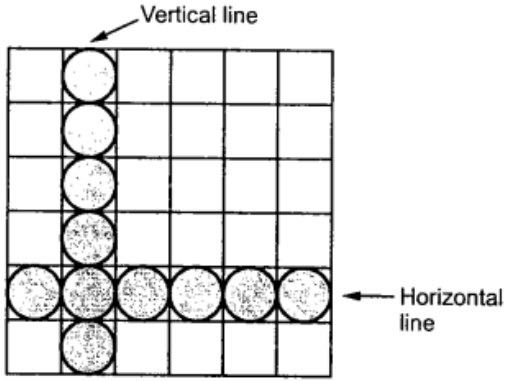


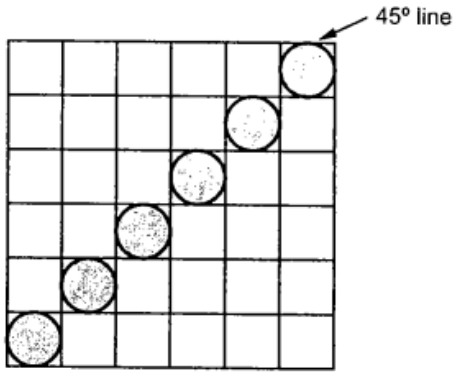
Fig: Uneven line density caused by bunching of dots.

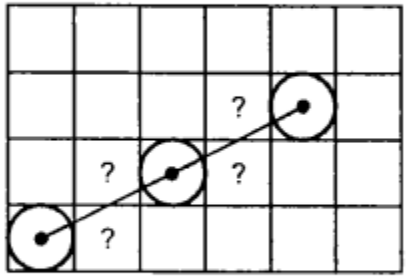
Algorithms for Line drawing

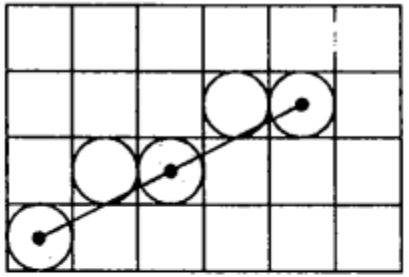
- Direct use of line equation.
- Digital Differential Analyzer (DDA).
- Bresenham’s Algorithm.

Basic concept of Line drawing

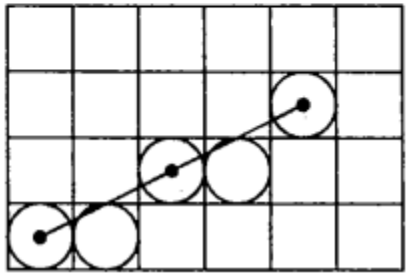








OR



Direct use of Line Equation

- It is the simplest form of conversion. First of all scan P_1 and P_2 points. P_1 has co-ordinates (x_1',y_1') and (x_2' y_2').
- Then m = (y_2',y_1')/(x_2',x_1') and $b = y_1^1 + mx_1^1$
- If value of |m|≤1 for each integer value of x. But do not consider x_1^1 and x_2^2
- If value of |m|>1 for each integer value of y. But do not consider y_1^1 and y_2^2

Algorithm for drawing line using equation

- **Step1:** Start Algorithm
- **Step2:** Declare variables x_1,x_2,y_1,y_2,dx,dy,m,b,
- **Step3:** Enter values of x_1,x_2,y_1,y_2.
  The (x_1,y_1) are co-ordinates of a starting point of the line.
  The (x_2,y_2) are co-ordinates of a ending point of the line.
- **Step4:** Calculate dx = x_2- x_1
- **Step5:** Calculate dy = y_2-y_1
- **Step6:** Calculate m = $\frac{dy}{dx}$
- **Step7:** Calculate b = y_1-m* x_1

- **Step8:** Set (x, y) equal to starting point, i.e., lowest point and x_{end}equal to largest value of x.
- If dx < 0
  then $x = x_2$
  $y = y_2$
  $x_{end} = x_1$
  If dx > 0
  then $x = x_1$
  $y = y_1$
  $x_{end} = x_2$
- **Step9:** Check whether the complete line has been drawn if $x=x_{end}$, stop
- **Step10:** Plot a point at current (x, y) coordinates
- **Step11:** Increment value of x, i.e., x = x+1
- **Step12:** Compute next value of y from equation y = mx + b
- **Step13:** Go to Step9.

- **Example:** A line with starting point as (0, 0) and ending point (6, 18) is given. Calculate value of intermediate points and slope of line.
- **Solution:** P_1 (0,0) P_7 (6,18)
- $x_1=0$
  $y_1=0$
  $x_2=6$
  $y_2=18$

- We know equation of line is
  $y = m x + b$
  $y = 3x + b$.....equation (1)
- put value of x from initial point in equation (1), i.e., (0, 0) x =0, y=0
  $0 = 3 \times 0 + b$
  $0 = b \Longrightarrow b=0$
- put b = 0 in equation (1)

- $y = 3x + 0$

  $y = 3x$
- Now calculate intermediate points

  Let $x = 1 \Longrightarrow y = 3 \times 1 \Longrightarrow y = 3$

  Let $x = 2 \Longrightarrow y = 3 \times 2 \Longrightarrow y = 6$

  Let $x = 3 \Longrightarrow y = 3 \times 3 \Longrightarrow y = 9$

  Let $x = 4 \Longrightarrow y = 3 \times 4 \Longrightarrow y = 12$

  Let $x = 5 \Longrightarrow y = 3 \times 5 \Longrightarrow y = 15$

  Let $x = 6 \Longrightarrow y = 3 \times 6 \Longrightarrow y = 18$
- So points are P_1 (0,0)

  P_2 (1,3)

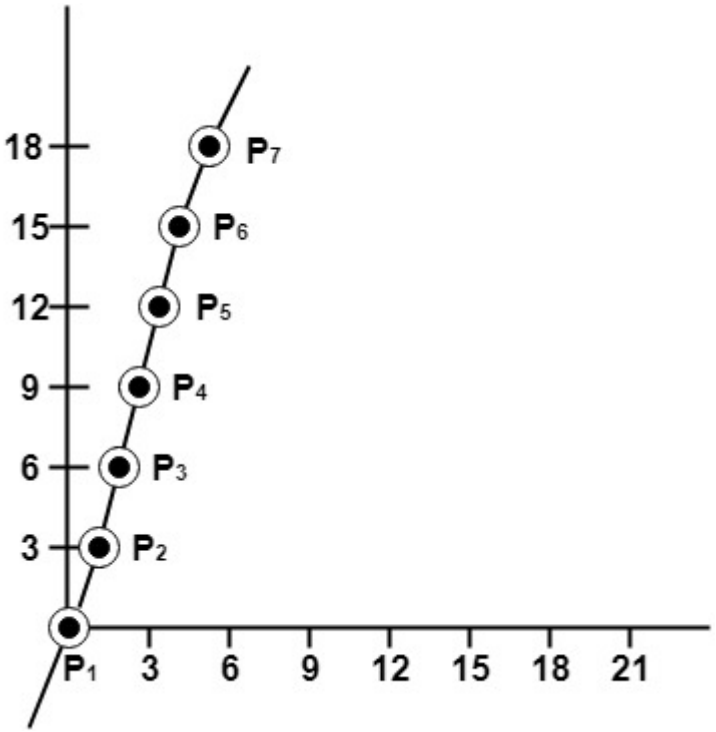
  P_3 (2,6)

  P_4 (3,9)

  P_5 (4,12)

  P_6 (5,15)

  P_7 (6,18)



Direct Differential Analyzer (DDA)

- DDA stands for Digital Differential Analyzer.
- It is an incremental method of scan conversion of line. In this method calculation is performed at each step but by using results of previous steps.
- The DDA is a scan-conversion line algorithm based on calculating either $^{\delta}y$ and $^{\delta}x$, using line equations.

DDA Algorithm

1. Read the line end points (x_1, y_1) and (x_2, y_2) such that they are not equal.
   [if equal then plot that point and exit]
2. $\Delta x = |x_2 - x_1|$ and $\Delta y = |y_2 - y_1|$
3. if $(\Delta x \geq \Delta y)$ then
   length = Δx
   else
   length = Δy
   end if
4. $\Delta x = (x_2 - x_1) / \text{length}$
   $\Delta y = (y_2 - y_1) / \text{length}$
   [This makes either Δx or Δy equal to 1 because length is either $|x_2 - x_1|$ or $|y_2 - y_1|$. Therefore, the incremental value for either x or y is one.]
5. $x = x_1 + 0.5 * \text{Sign}(\Delta x)$
   $y = y_1 + 0.5 * \text{Sign}(\Delta y)$
   [Here, Sign function makes the algorithm work in all quadrant. It returns - 1, 0, 1 depending on whether its argument is < 0, = 0, > 0 respectively. The factor 0.5 makes it possible to round the values in the integer function rather than truncating them.]
6. i = 1 [Begins the loop, in this loop points are plotted]
   While (i ≤ length)
   {
   Plot (Integer (x), Integer (y))
   $x = x + \Delta x$
   $y = y + \Delta y$
   i = i + 1
7. Stop

Advantages of DDA

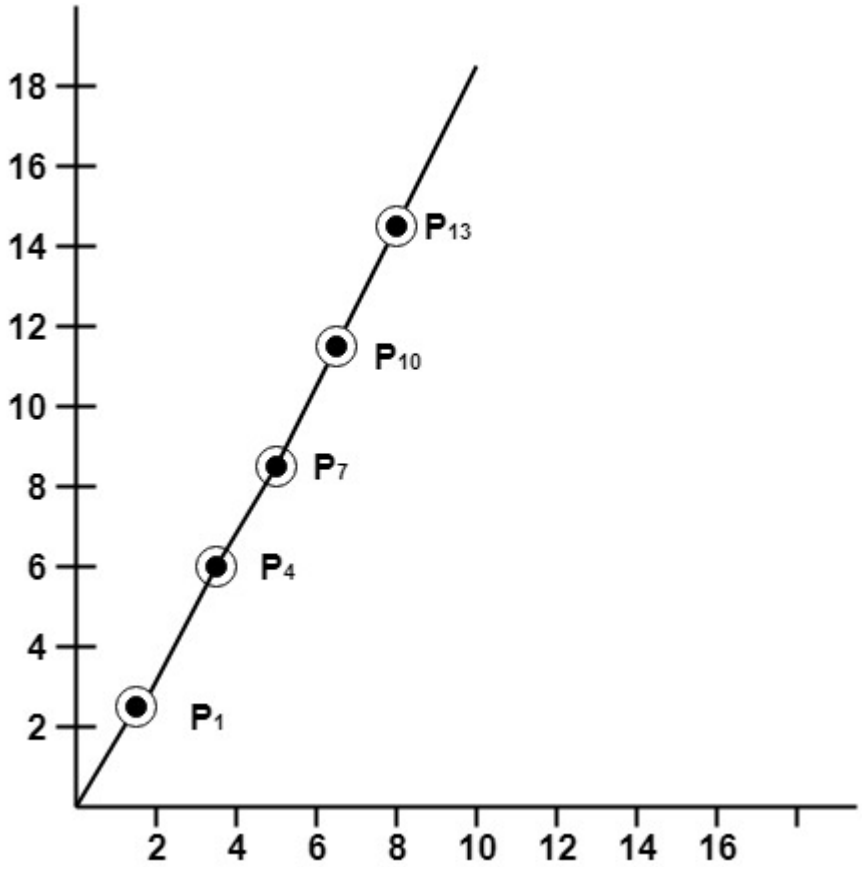
- Advantages:
  - It is a faster method than method of using direct use of line equation.
  - This method does not use multiplication theorem.
  - It allows us to detect the change in the value of x and y, so plotting of same point twice is not possible.
  - This method gives overflow indication when a point is repositioned.
  - It is an easy method because each step involves just two additions.

Disadvantages of DDA

- Disadvantages:
  - It involves floating point additions rounding off is done. Accumulations of round off error cause accumulation of error.
  - Rounding off operations and floating point operations consumes a lot of time.
  - It is more suitable for generating line using the software. But it is less suited for hardware implementation.

- **Example:** If a line is drawn from (2, 3) to (6, 15) with use of DDA. How many points will needed to generate such line?
- **Solution:** P_1 (2,3) P_{11} (6,15)
- $x_1=2$
  $y_1=3$
  $x_2= 6$
  $y_2=15$
  $dx = 6 - 2 = 4$
  $dy = 15 - 3 = 12$
  $m = 3$
- For calculating next value of x takes $x = x + 1/m$
-

$P_1(2, 3)$	point plotted
$P_2(2\frac{1}{3}, 4)$	point plotted
$P_3(2\frac{2}{3}, 5)$	point not plotted
$P_4(3, 6)$	point plotted
$P_5(3\frac{1}{3}, 7)$	point not plotted
$P_6(3\frac{2}{3}, 8)$	point not plotted
$P_7(4, 9)$	point plotted
$P_8(4\frac{1}{3}, 10)$	point not plotted
$P_9(4\frac{2}{3}, 11)$	point not plotted
$P_{10}(5, 12)$	point plotted
$P_{11}(5\frac{1}{3}, 13)$	point not plotted
$P_{12}(5\frac{2}{3}, 14)$	point not plotted
$P_{13}(6, 15)$	point plotted



Problems of DDA

- Consider the line from (0,0) to (4,6). Use the simple DDA algorithm to rasterize this line segment.

- Consider the line from (0,0) to (-6,-6). Use the simple DDA algorithm to rasterize this line segment.

Bresenham's Line Drawing Algorithm

- This algorithm is used for scan converting a line.
- It was developed by Bresenham.
- It is an efficient method because it involves only integer addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly.
- In this method, next pixel selected is that one who has the least distance from true line.

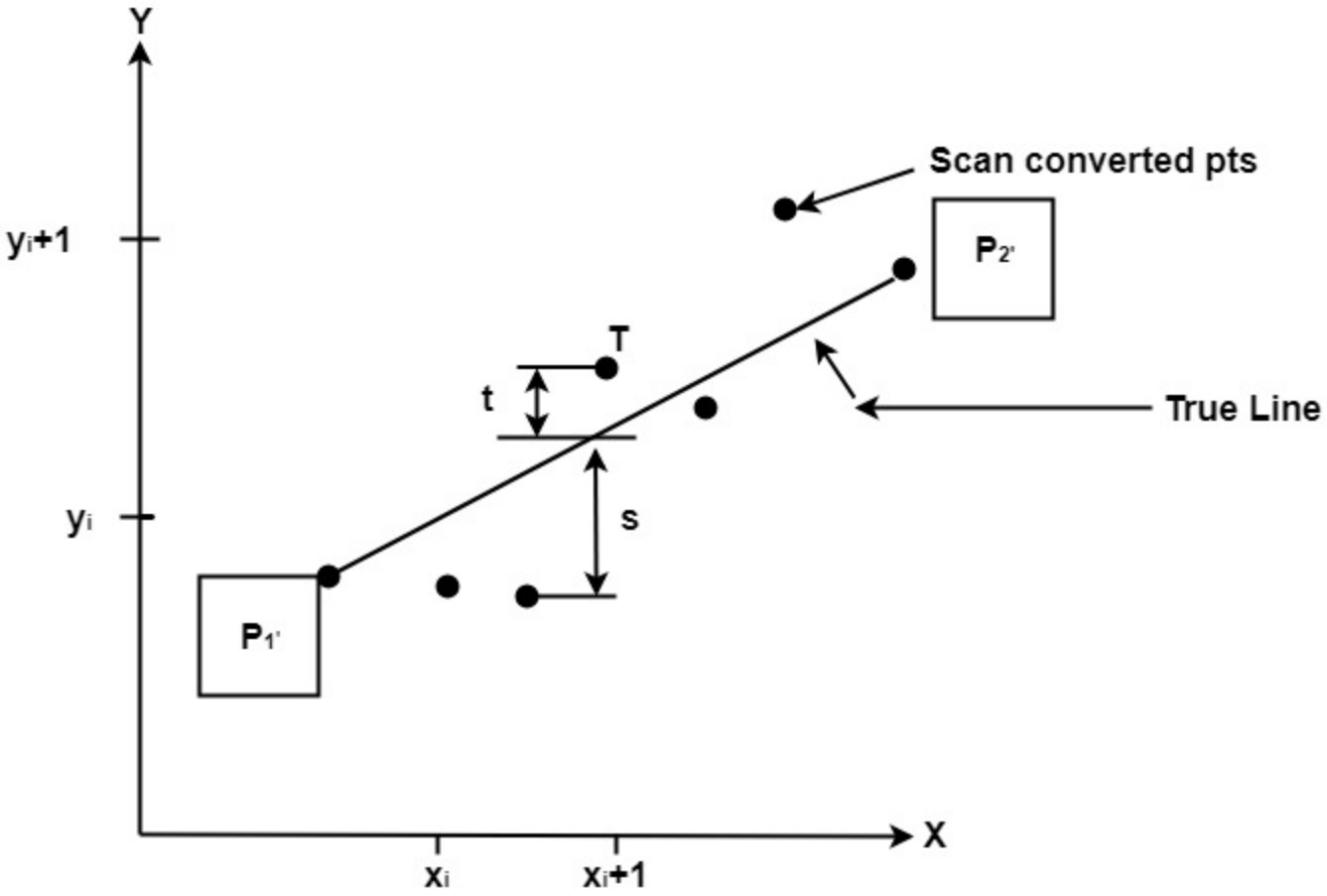
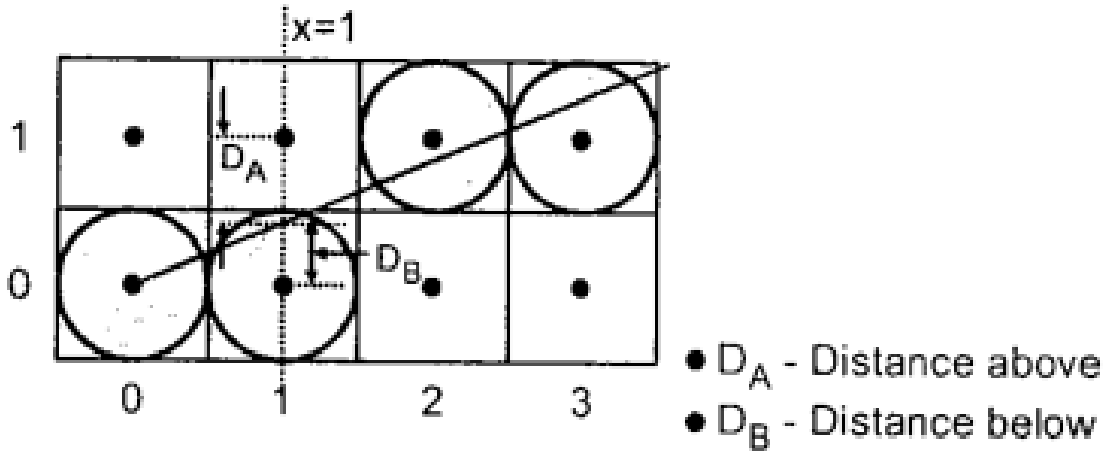


Fig: Scan Converting a line.


x=1
1
0
D_A
D_B
0
1
2
3
• D_A - Distance above
• D_B - Distance below

Bresenham's Algorithm

1. Read the line end points (x_1, y_1) and (x_2, y_2) such that they are not equal.
   [if equal then plot that point and exit]
2. $\Delta x = |x_2 - x_1|$ and $\Delta y = |y_2 - y_1|$
3. [Initialize starting point]
   $x = x_1$
   $y = y_1$
4. $e = 2 * \Delta y - \Delta x$
   [Initialize value of decision variable or error to compensate for nonzero intercepts]
5. i = 1 [Initialize counter]
6. Plot (x, y)
7. while ($e \geq 0$)
   {
   $y = y + 1$
   $e = e - 2 * \Delta x$
   }
   $x = x + 1$
   $e = e + 2 * \Delta y$
8. $i = i + 1$
9. if ($i \leq \Delta x$) then go to step 6.
10. Stop

Advantages of Bresenham's Algorithm

- Advantages:
  - It involves only integer arithmetic, so it is simple.
  - It avoids the generation of duplicate points.
  - It can be implemented using hardware because it does not use multiplication and division.
  - It is faster as compared to DDA (Digital Differential Analyzer) because it does not involve floating point calculations like DDA Algorithm.

Disadvantages of Bresenham’s Algorithm

- Disadvantages:
  - This algorithm is meant for basic line drawing only Initializing is not a part of Bresenham's line algorithm. So to draw smooth lines, you should want to look into a different algorithm.

- **Example:** Starting and Ending position of the line are (1, 1) and (8, 5). Find intermediate points.
- **Solution:** $x_1=1$

$y_1=1$

$x_2=8$

$y_2=5$

$dx= x_2-x_1=8-1=7$

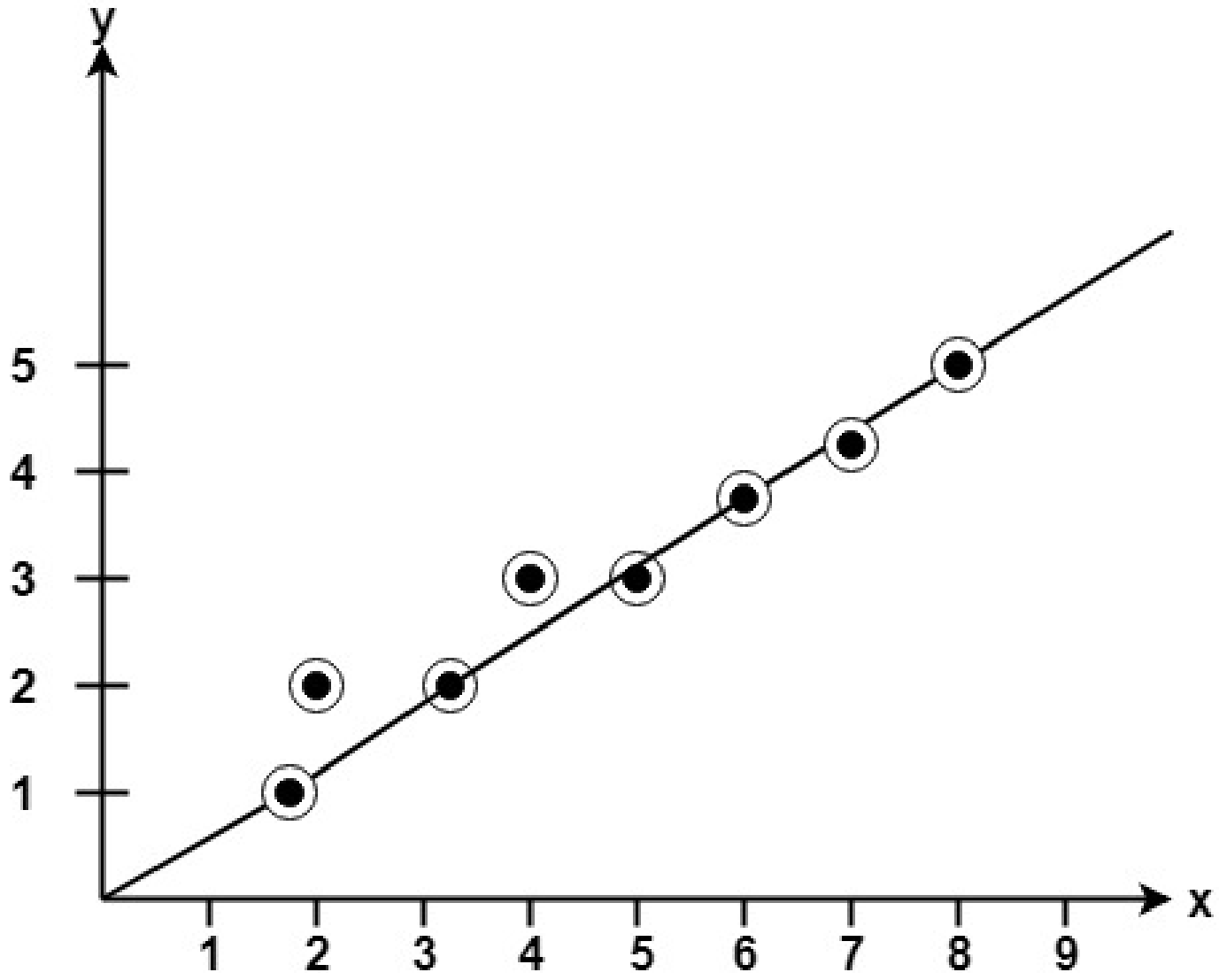
$dy=y_2-y_1=5-1=4$

$I_1=2* \Delta y=2*4=8$

$I_2=2*(\Delta y-\Delta x)=2*(4-7)=-6$

$d = I_1-\Delta x=8-7=1$

x	y	d=d+I_1 or I_2
1	1	d+I_2=1+(-6)=-5
2	2	d+I_1=-5+8=3
3	2	d+I_2=3+(-6)=-3
4	3	d+I_1=-3+8=5
5	3	d+I_2=5+(-6)=-1
6	4	d+I_1=-1+8=7
7	4	d+I_2=7+(-6)=1
8	5	


y
5
4
3
2
1
x
1
2
3
4
5
6
7
8
9

Difference Between DDA and Bresenham's Line Algorithms

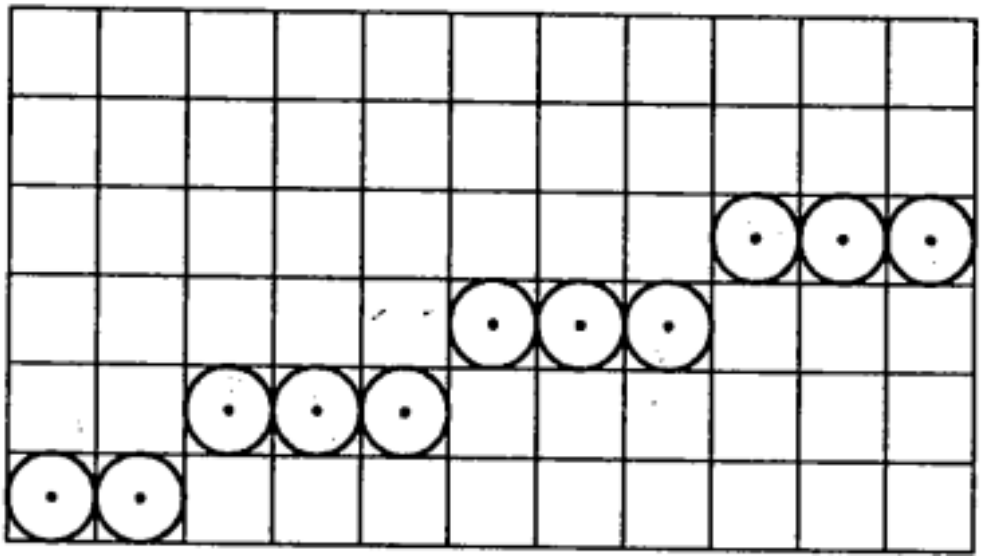
DDA Algorithm	Bresenham's Line Algorithm
1. DDA Algorithm use floating point, i.e., Real Arithmetic.	1. Bresenham's Line Algorithm use fixed point, i.e., Integer Arithmetic
2. DDA Algorithms uses multiplication & division its operation	2.Bresenham's Line Algorithm uses only subtraction and addition its operation
3. DDA Algorithm is slowly than Bresenham's Line Algorithm in line drawing because it uses real arithmetic (Floating Point operation)	3. Bresenham's Algorithm is faster than DDA Algorithm in line because it involves only addition & subtraction in its calculation and uses only integer arithmetic.
4. DDA Algorithm is not accurate and efficient as Bresenham's Line Algorithm.	4. Bresenham's Line Algorithm is more accurate and efficient at DDA Algorithm.
5.DDA Algorithm can draw circle and curves but are not accurate as Bresenham's Line Algorithm	5. Bresenham's Line Algorithm can draw circle and curves with more accurate than DDA Algorithm.

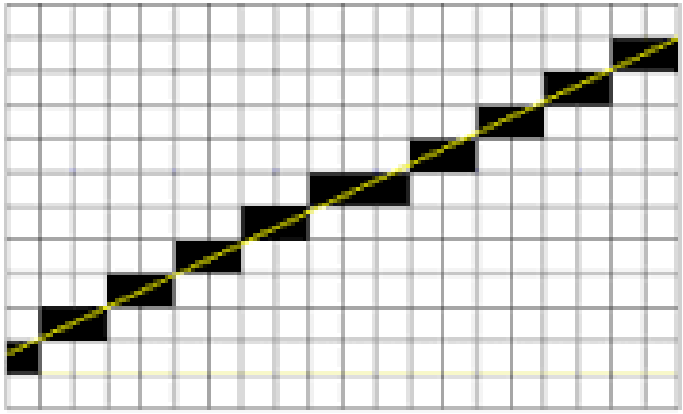
Antialiasing of Lines

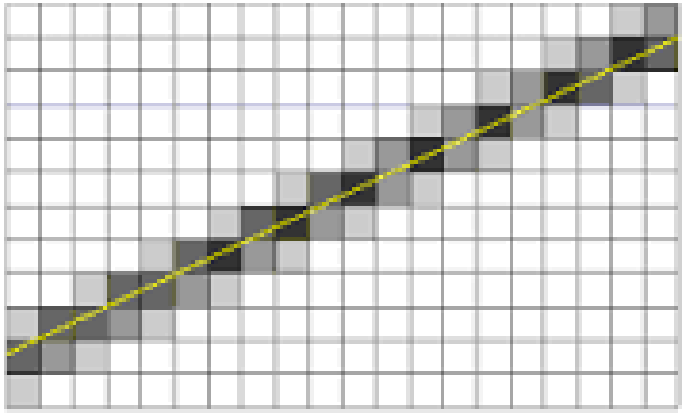
- In the line drawing algorithm, we have seen that all rasterized locations do not match with the true line and we have to select the optimum raster locations to represent a straight line.

- This problem is severe in low resolution screens. In such screens line appears like a stair-step.

- The aliasing effect is the appearance of jagged edges or “jaggies” in a rasterized image (an image rendered using pixels).



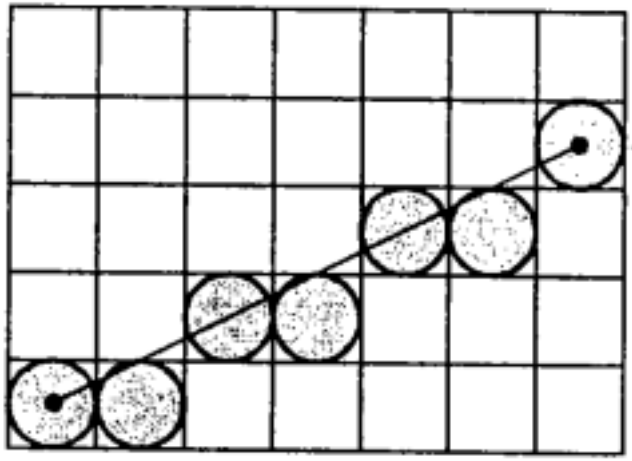


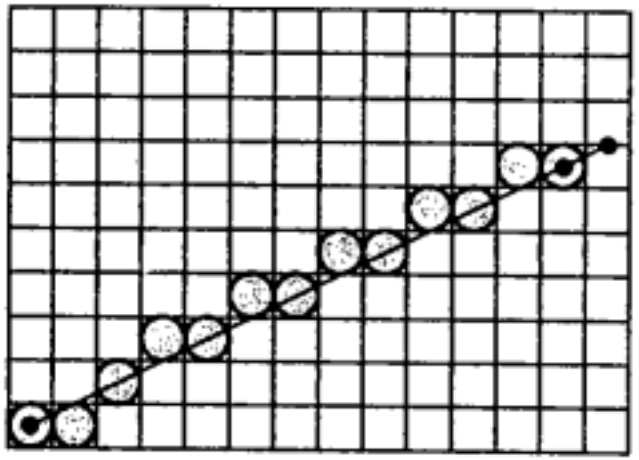


- The problem of jagged edges technically occurs due to distortion of the image when scan conversion is done with sampling at a low frequency, which is also known as Undersampling.
- Aliasing occurs when real-world objects which comprise of smooth, continuous curves are rasterized using pixels.
- **Antialiasing** is a technique used in computer graphics to remove the aliasing effect.
- The aliasing effect can be reduced by adjusting intensities of the pixels along the line.
- The process of adjusting intensities of the pixels along the line to minimize the effect of aliasing is called antialiasing.

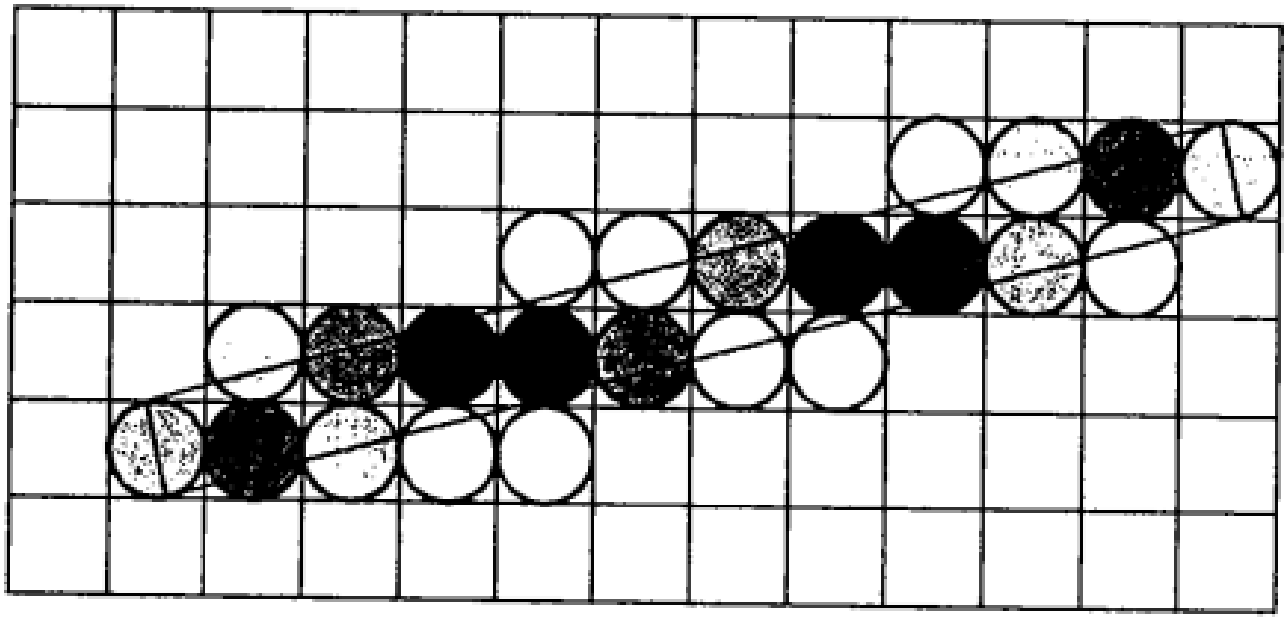
Methods of Antialiasing:

- **Increasing resolution:**

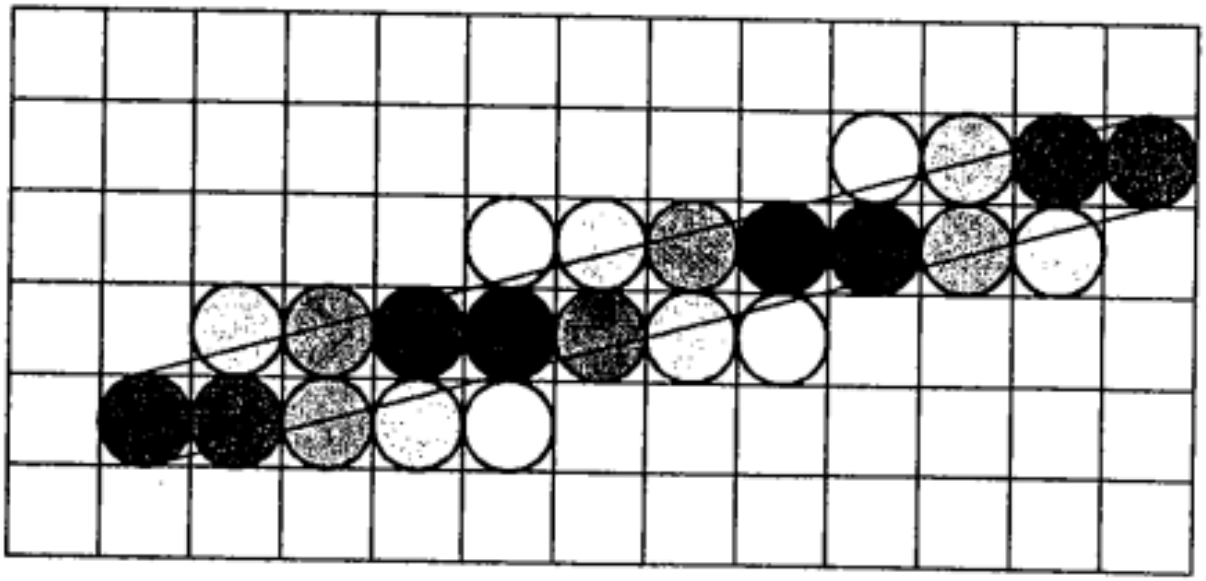




- **Unweighted area sampling:** The intensity of pixel is proportional to the amount of line area occupied by the pixel. This technique produces noticeably better results than does setting pixels either to full intensity or to zero intensity.



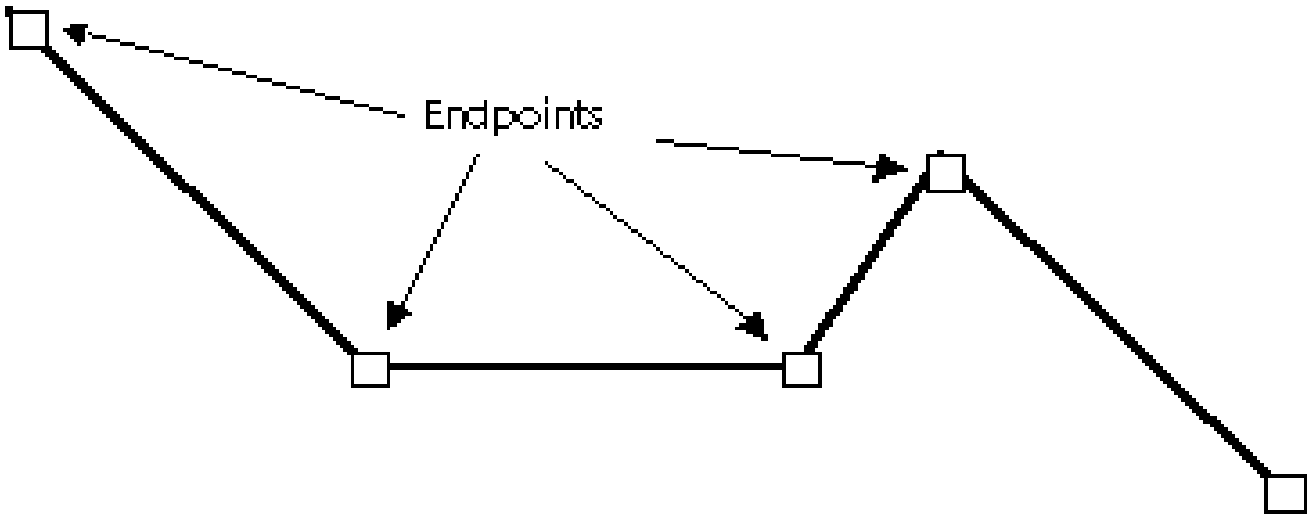
- **Weighted area sampling:** in weighted area sampling equal areas contribute unequally i.e. a small area closer to the pixel center has greater intensity than does one at a greater distance. Thus, the intensity of the pixel is dependent on the line area occupied and the distance of area from the pixel's center.



Displaying Polylines

- Implementation of a polyline procedure is accomplished by invoking a line-drawing routine *n-1* times to display the line connecting the *n* endpoints.
- Each successive call passes the coordinate pair needed to plot the next line section, where the first endpoint of each coordinate pair is the last endpoint of the previous section.
- Once the color values for pixel positions along the first line segment have been set in the frame buffer, we process subsequent line segments starting with the next pixel position following the first endpoint for that segment.
- In this way, we can avoid setting the color of some endpoints twice.

Polyline representation



Parallel Line Algorithms

- Using parallel processing, we can calculate multiple pixel positions along a line path simultaneously by partitioning the computations among the various processors available.
- One approach to the partitioning problem is to adapt an existing sequential algorithm to take advantage of multiple processors.
- Alternatively, to set up the processing so that pixel positions can be calculated efficiently in parallel.
- An important consideration in devising a parallel algorithm is to balance the processing load among the available processors.

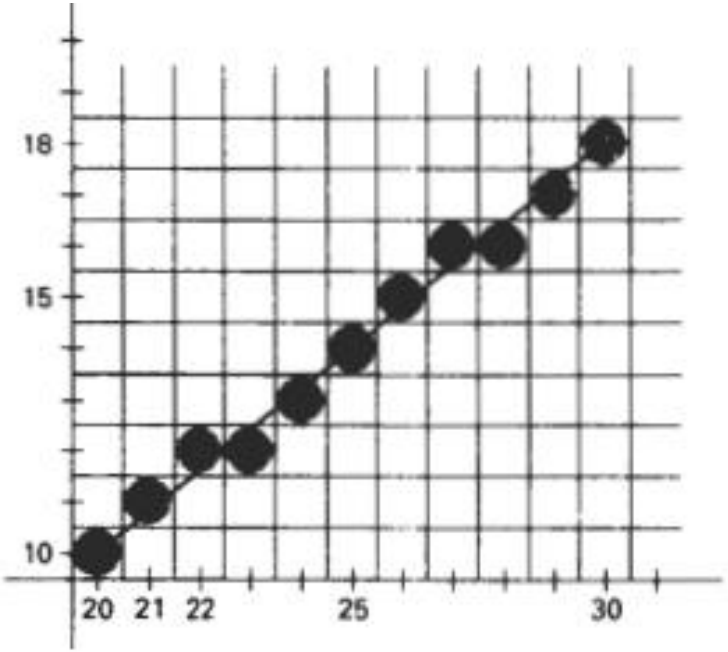


Figure 3-9
Pixel positions along the line path between endpoints (20, 10) and (30, 18), plotted with Bresenham's line algorithm.

- Given n_p processors, we can set up a parallel Bresenham line algorithm by subdividing the line path into n_p partitions and simultaneously generating line segments in each of the subintervals.
- For a line with slope $0 < m < 1$ and left endpoint coordinate position (x0, y0), we partition the line along the positive x direction.
- The distance between beginning x positions of adjacent partitions can be calculated as:

$$\Delta x_p = \frac{\Delta x + n_p - 1}{n_p}$$

- Numbering the partitions, and the processors, as 0,1,2, up to n_p-1, we calculate the starting x coordinate for the *k*th partition as: $x_k = x_0 + k\Delta x_p$

Scan Conversion Circle

- A circle is defined as the set of points that are all at a given distance r from a center position.
- Circle is an eight-way symmetric figure.
- The shape of circle is the same in all quadrants. In each quadrant, there are two octants.
- If the calculation of the point of one octant is done, then the other seven points can be calculated easily by using the concept of eight-way symmetry.

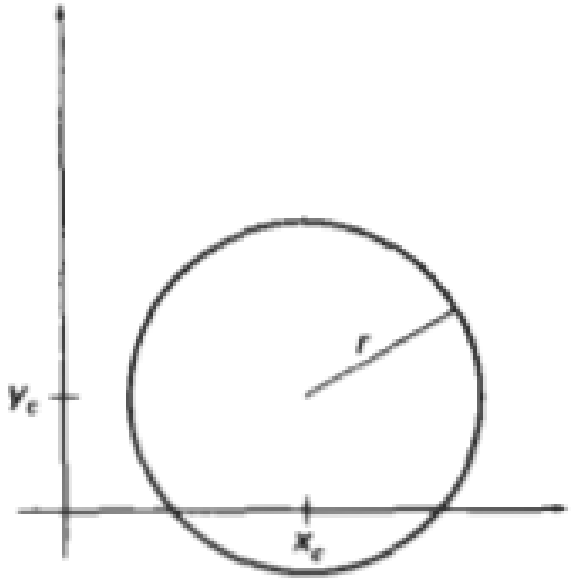
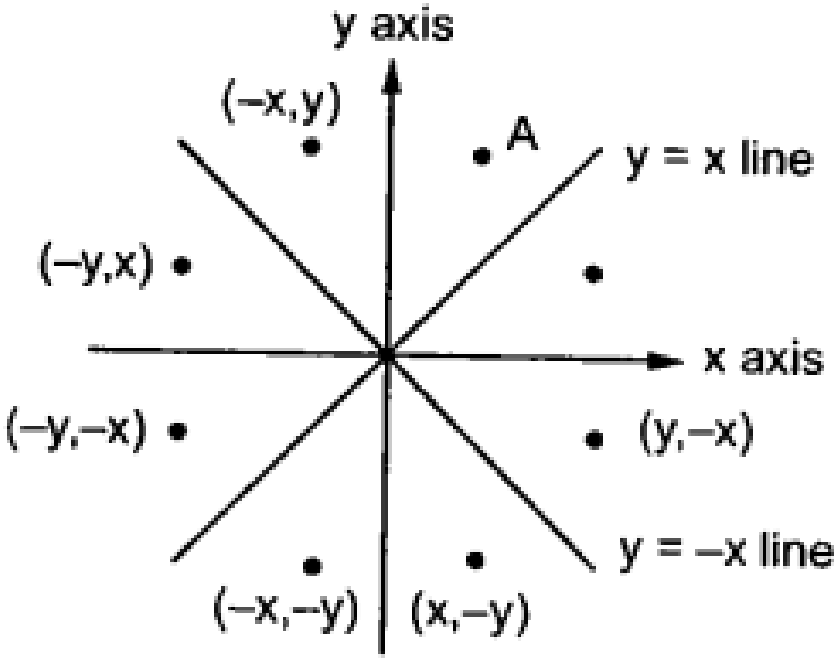


Figure 3-12
Circle with center coordinates (x_c, y_c) and radius r.



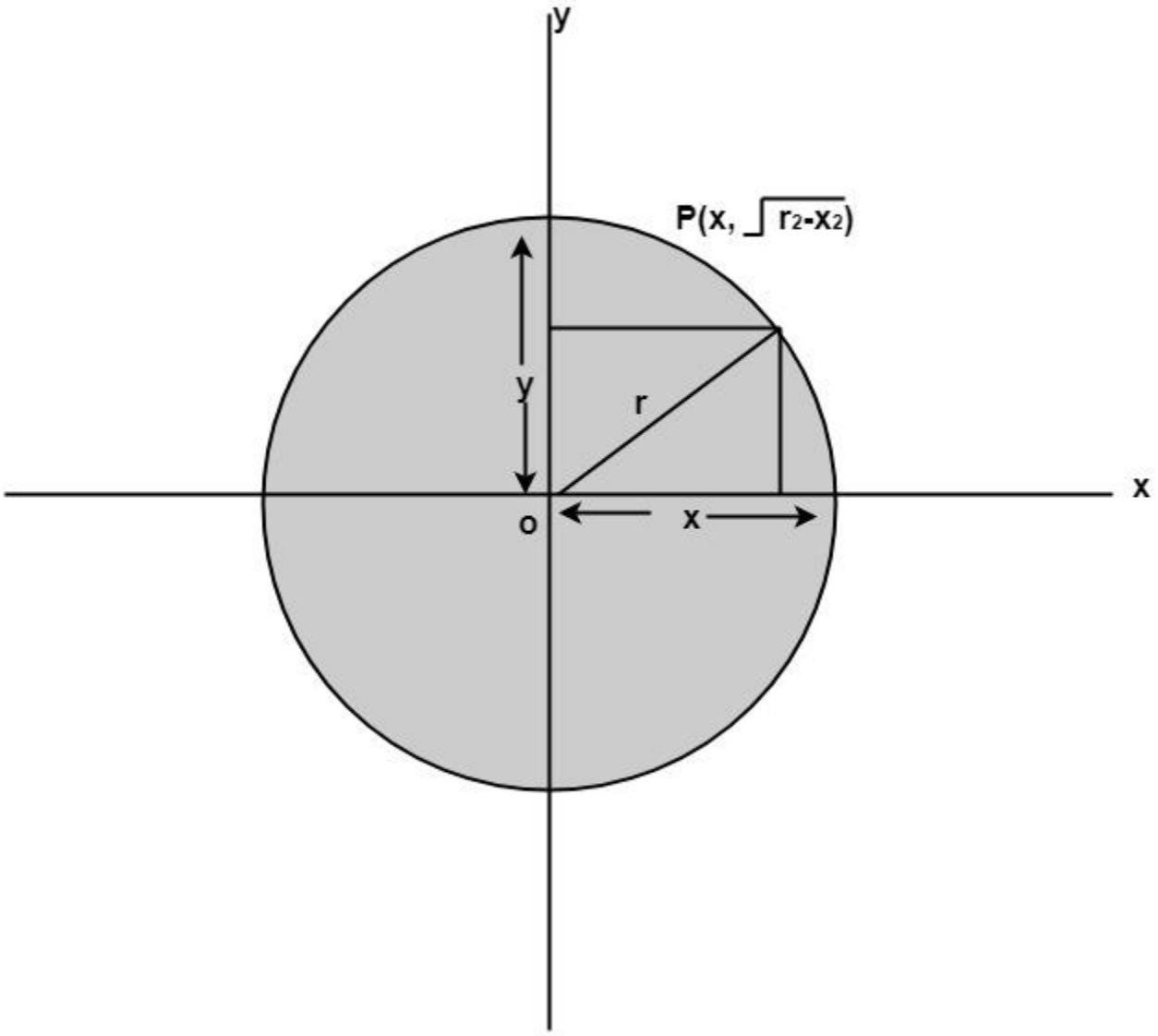
Defining circle using Polynomial method

- The first method defines a circle with the second-order polynomial equation as shown in fig:
- $y^2=r^2-x^2$

  Where x = the x coordinate

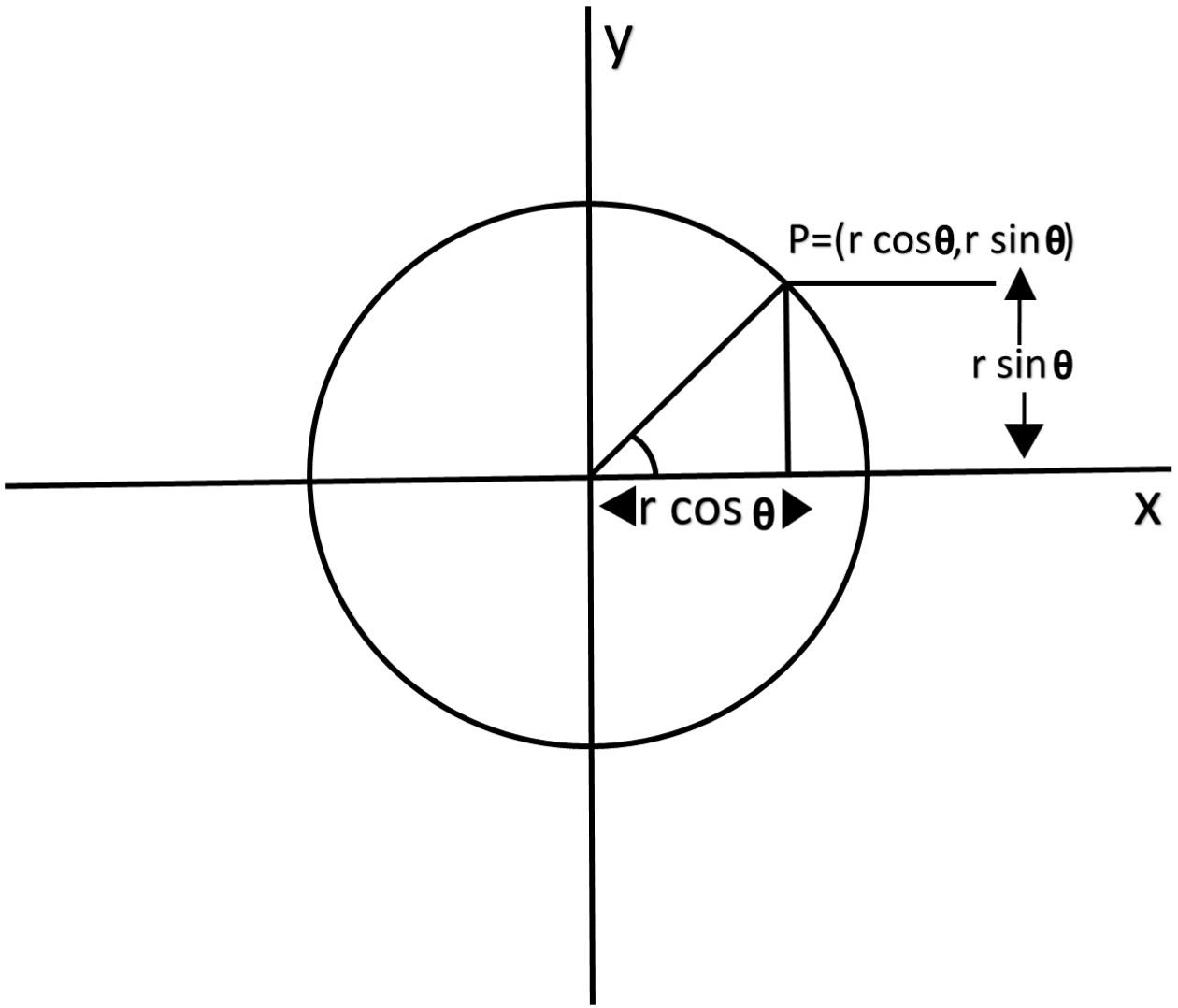
  y = the y coordinate

  r = the circle radius
- With the method, each x coordinate in the sector, from 90° to 45°, is found by stepping x from 0 to $r/\sqrt{2}$ & each y coordinate is found by evaluating $\sqrt{r^2-x^2}$ for each step of x.


y
P(x, √r2-x2)
y
r
x
o
x

Defining circle using Polar Coordinates

- The second method of defining a circle makes use of polar coordinates as shown in fig:
- $x = r \cos \theta$ $y = r \sin \theta$
  Where θ=current angle
  r = circle radius
  x = x coordinate
  y = y coordinate
- By this method, θ is stepped from 0 to $\pi/4$ & each value of x & y is calculated.


y
P=(r cosθ,r sinθ)
r sinθ
r cosθ
x

DDA Circle Drawing Algorithm

- The equation of circle, with origin as the center of the circle is given as: $x^2 + y^2 = r^2$
- The DDA algorithm can be used to draw the circle by defining circle as a differential equation.

$$2x\,dx + 2y\,dy = 0 \quad \text{where} \quad r \quad \text{is constant}$$

$$\therefore \quad x\,dx + y\,dy = 0$$

$$\therefore \quad y\,dy = -x\,dx$$

$$\therefore \quad \frac{dy}{dx} = \frac{-x}{y}$$

DDA Circle Drawing Algorithm…

1. Read the radius (r), of the circle and calculate value of ε
2. start_x = 0

   start_y = r
3. x_1 = start_x

   y_1 = start_y
4. do

   { $x_2 = x_1 + \varepsilon y_1$

   $y_2 = y_1 - \varepsilon x_2$

   [x_2 represents x_{n+1} and x_1 represents x_n]

   Plot (int (x_2) , int (y_2))

   $x_1 = x_2$;

   $y_1 = y_2$;

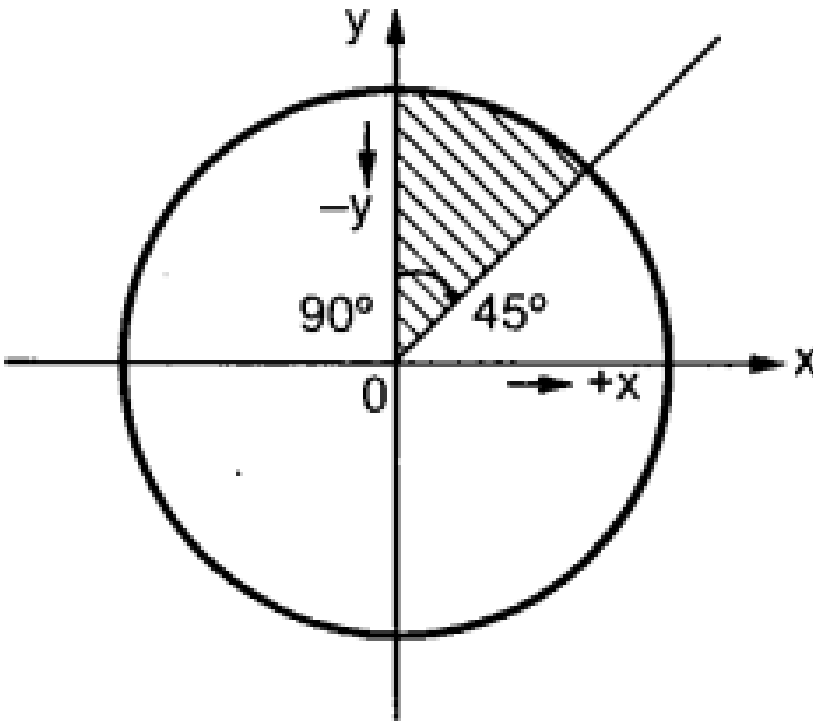
   [Reinitialize the current point]

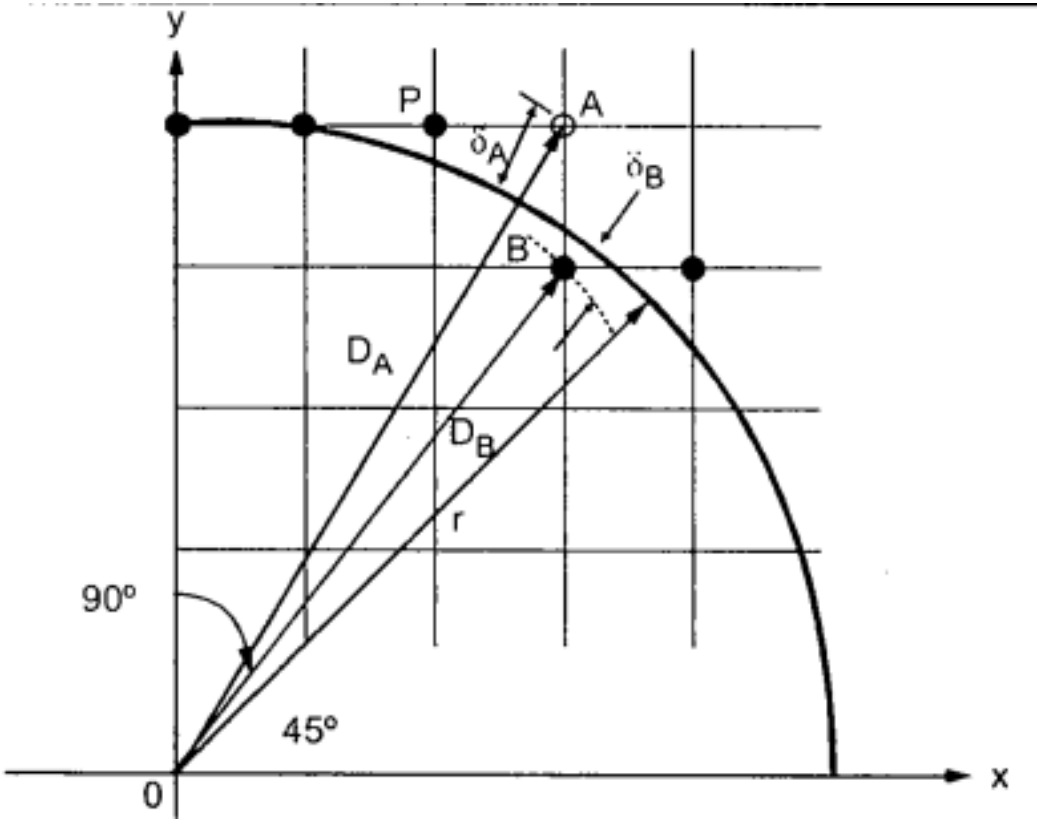
   } while (y_1 – start_y) < ε or (start_x – x_1) > ε

   [check if the current point is the starting point or not. If current point is not starting point repeat step 4 ; otherwise stop]
5. Stop.

Bresenham's Circle Algorithm

- The Bresenham's circle drawing algorithm considers the eight-way symmetry of the circle to generate it.
- It plots $1/8^{th}$ part of the circle, i.e. from 90 degree to 45 degree.
- As circle is drawn from 90 degree to 45 degree, the x moves in positive direction and y moves in the negative direction.


y
–y
90°
45°
0
+x
x


y
P
A
δA
δB
B
DA
DB
r
90°
45°
0
x

- To achieve best approximation to the true circle we have to select those pixels in the raster that fall the least distance from the true circle.
- Let us observe the 90 degree to 45 degree portion of the circle.
- It can be noticed that if points are generated from 90 degree to 45 degree, each point closest to the true circle can be found by applying either of the two options:
  - Increment in positive x direction by one unit or
  - Increment in positive x direction and negative y direction both by one unit.

Bresenham's algo. to plot 1/8th of the circle

1. Read the radius (r) of the circle.
2. d = 3 – 2 r

   [Initialize the decision variable]
3. x = 0, y = r

   [Initialize starting point]
4. do

   {

   plot (x, y)

   if (d < 0) then

   {

   d = d + 4x + 6

   }

   else

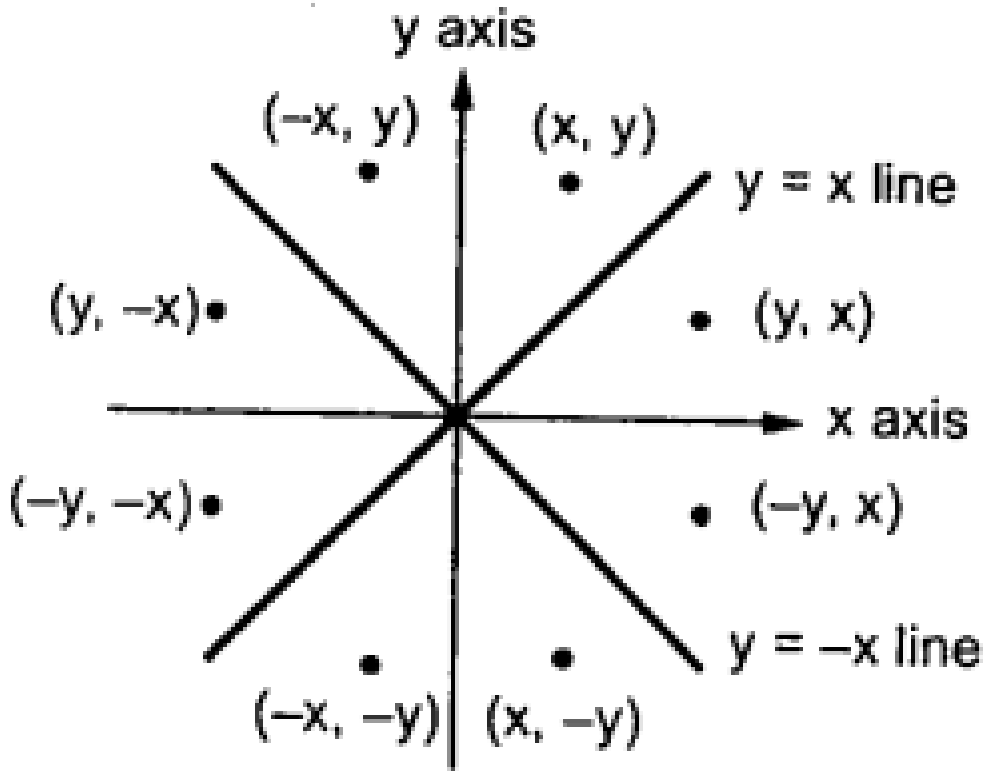
   { d = d + 4 (x – y) + 10

   y = y – 1

   }

   x = x + 1

   } while (x < y)
5. Stop



plot (y, x)

plot (y, – x)

plot (x, – y)

plot (– x, – y)

plot (– y, – x)

plot (– y, x) and

plot (– x, y)

Midpoint circle drawing algorithm

- The midpoint circle drawing algorithm also uses the eight-way symmetry of the circle to generate it.
- As in the raster line algorithm, we sample at unit intervals and determine the closest pixel position to the specified circle path at each step.
- For a given radius r and screen center position (xc, yc), we can first set up our algorithm to calculate pixel positions around a circle path centered at the coordinate origin (0, 0).
- Then each calculated position (x, y) is moved to its proper screen position by adding xc to x and yc to y.

$$f_{\text{circle}}(x, y) = x^2 + y^2 - r^2$$

$$f_{\text{circle}}(x, y) \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the circle boundary} \\ = 0, & \text{if } (x, y) \text{ is on the circle boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the circle boundary} \end{cases}$$

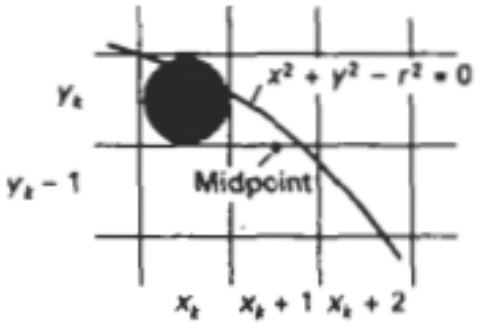


Figure 3-15
Midpoint between candidate pixels at sampling position x_k+1 along a circular path.

$$p_k = f_{\text{circle}}\left(x_k + 1, y_k - \frac{1}{2}\right)$$

$$= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$$

$$p_{k+1} = f_{\text{circle}}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$

$$= [(x_k + 1) + 1]^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2$$

$$p_{k+1} = p_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

$$2x_{k+1} = 2x_k + 2$$

$$2y_{k+1} = 2y_k - 2$$

Midpoint circle drawing algorithm…

1. Input radius r and circle center (x_c, y_c), and obtain the first point on the circumference of a circle centered on the origin as

$$(x_0, y_0) = (0, r)$$

2. Calculate the initial value of the decision parameter as

$$p_0 = \frac{5}{4} - r$$

3. At each x_k position, starting at $k = 0$, perform the following test: If $p_k < 0$, the next point along the circle centered on $(0, 0)$ is (x_{k+1}, y_k) and

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ and

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

where $2x_{k+1} = 2x_k + 2$ and $2y_{k+1} = 2y_k - 2$.

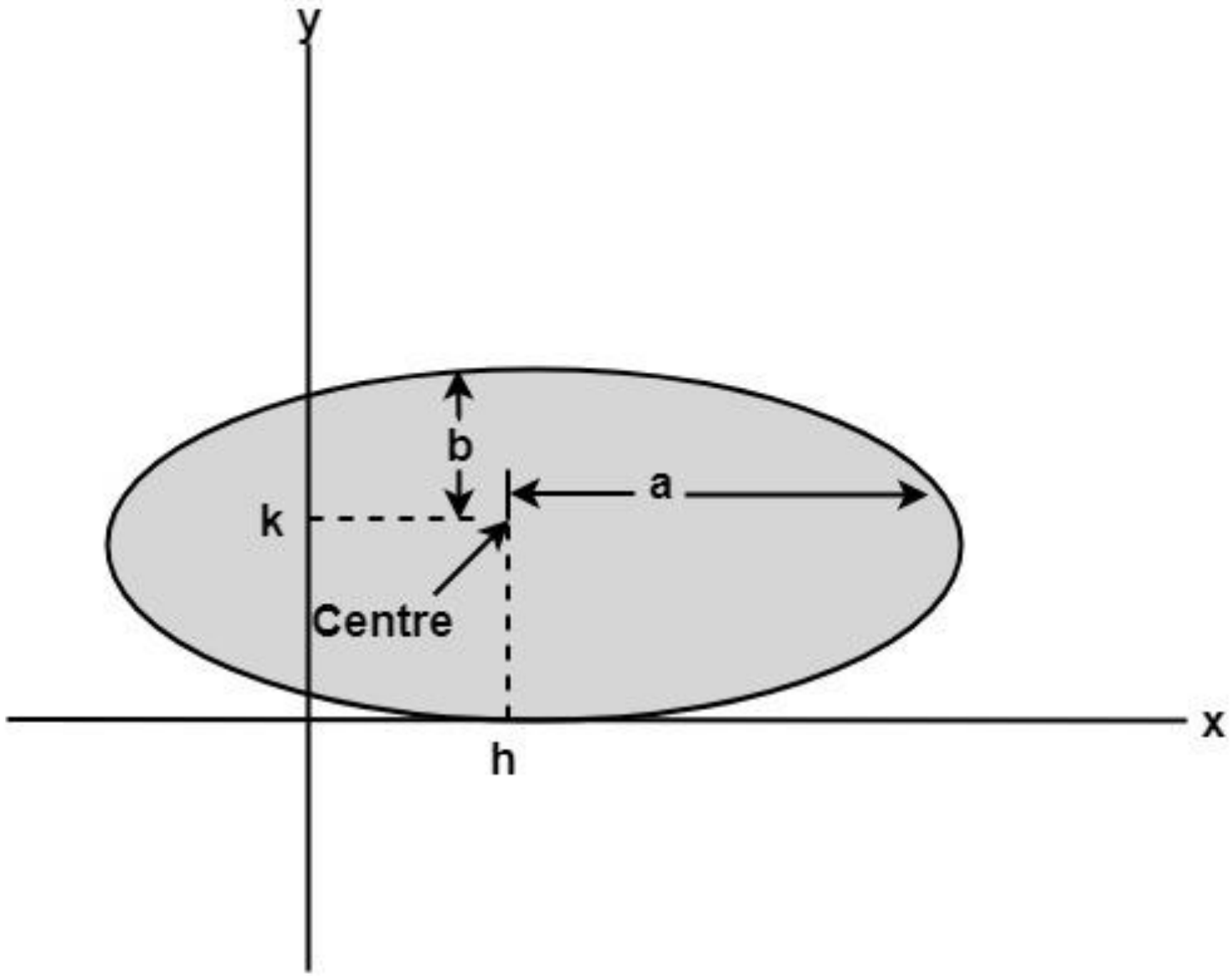
4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot the coordinate values:

$$x = x + x_c, \qquad y = y + y_c$$

6. Repeat steps 3 through 5 until $x \geq y$.

Scan conversion of Ellipse

- An ellipse is an elongated circle.
- Therefore, elliptical curves can be generated by modifying circle-drawing procedures to make into account the different dimensions of an ellipse along the major and minor axes.
- The ellipse is also a symmetric figure like a circle but is four-way symmetry rather than eight-way.
- Two methods of defining an Ellipse:
  - Polynomial method
  - Trigonometric method


y
b
a
k
Centre
x
h

Polynomial method for Ellipse

- The ellipse has a major and minor axis. If a_1 and b_1 are major and minor axis respectively. The centre of ellipse is (i, j). The value of x will be incremented from i to a_1 and value of y will be calculated using the following formula

$$y = b_1 \sqrt{1 - \frac{x - i}{a_1^2}} + j$$

Drawback of Polynomial method:

- It requires squaring of values. So floating point calculation is required.
- Routines developed for such calculations are very complex and slow.

Trigonometric method for Ellipse

- The following equation defines an ellipse trigonometrically as shown in fig:

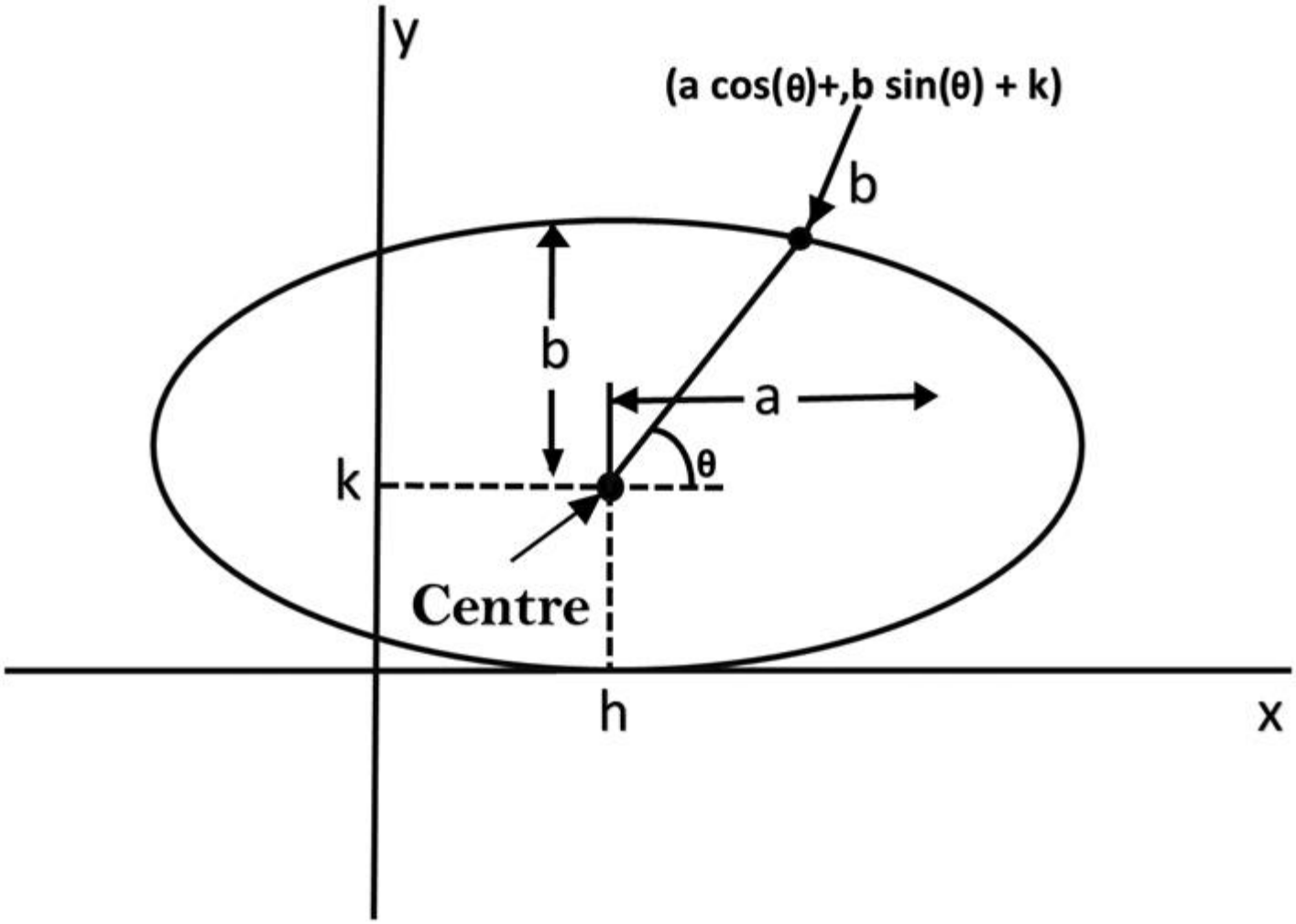
  $x = a * \cos(\theta) + h$ and
  $y = b * \sin(\theta) + k$

  where (x, y) = the current coordinates
  a = length of major axis
  b = length of minor axis
  θ= current angle
  (h, k) = ellipse center

- In this method, the value of θ is varied from 0 to $\pi/2$ radians. The remaining points are found by symmetry.

Drawback of Trigonometric method:

- This is an inefficient method.
- It is not an interactive method for generating ellipse.
- The table is required to see the trigonometric value.
- Memory is required to store the value of θ.


y
(a cos(θ)+,b sin(θ) + k)
b
b
a
θ
k
Centre
h
x

Properties of Ellipses

- An ellipse is defined as the set of points such that the sum of the distances from two fixed positions (foci) is the same for all points.
- If the two foci from any point P = (x, y) on the ellipse are labeled d1 and d2, then the general equation of an ellipse can be stated as:

$$d_1 + d_2 = \text{constant}$$

$$\sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2} = \text{constant}$$

- By squaring the last equation, isolating the remaining radical, and squaring again, we can rewrite the general ellipse equation in the form as:

$$A x^2 + B y^2 + C x y + D x + E y + F = 0$$

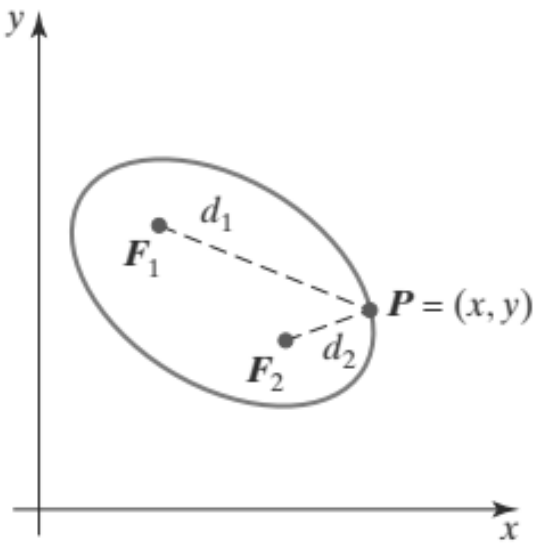


FIGURE 16
Ellipse generated about foci F_1 and F_2.

- Ellipse equations are greatly simplified if the major and minor axes are oriented to align with the coordinate axes.
- The equation for the ellipse shown in the figure can be written in terms of the ellipse center coordinates and parameter rx and ry as:

$$\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{y - y_c}{r_y}\right)^2 = 1$$

$$x = x_c + r_x \cos\theta$$

$$y = y_c + r_y \sin\theta$$

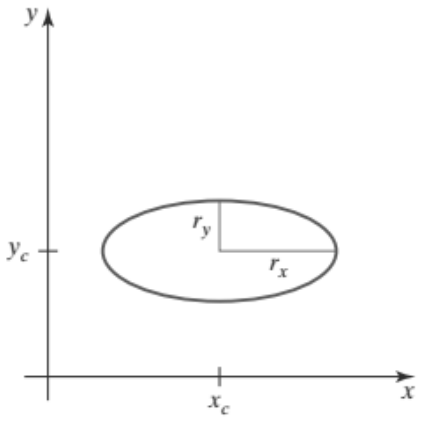


FIGURE 17
Ellipse centered at (x_c, y_c) with semimajor axis r_x and semiminor axis r_y.

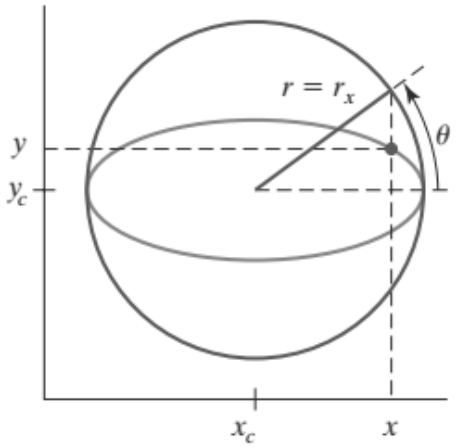


FIGURE 18
The bounding circle and eccentric angle θ for an ellipse with $r_x > r_y$.

Midpoint Ellipse Algorithm

- This is an incremental method for scan converting an ellipse that is centered at the origin in standard position i.e., with the major and minor axis parallel to coordinate system axis.
- It is very similar to the midpoint circle algorithm. Because of the four-way symmetry property we need to consider the entire elliptical curve in the first quadrant.

- The midpoint ellipse method is applied throughout the first quadrant in two parts.
- Regions 1 and 2 can be processed in various ways. We can start at position (0, ry) and step clockwise along the elliptical path in the first quadrant, shifting from unit steps in x to unit steps in y when slop< -1.0.

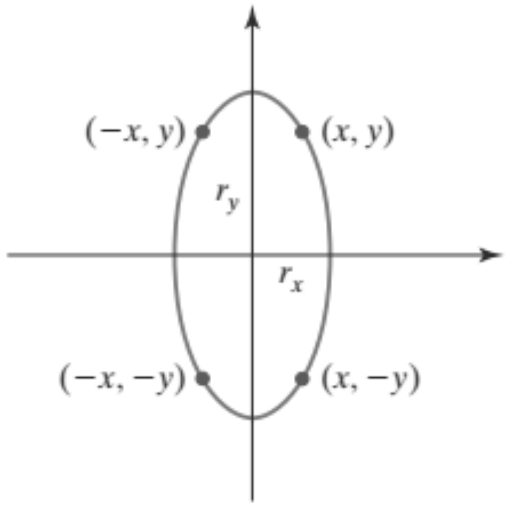


FIGURE 19
Symmetry of an ellipse. Calculation of a point (x, y) in one quadrant yields the ellipse points shown for the other three quadrants.

$$f_{\text{ellipse}}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2$$

$$f_{\text{ellipse}}(x, y) \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the ellipse boundary} \\ = 0, & \text{if } (x, y) \text{ is on the ellipse boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the ellipse boundary} \end{cases}$$

- At each step we need to test the value of the slope of the curve.
- The ellipse slope is calculated from equation as:

$$\frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y}$$

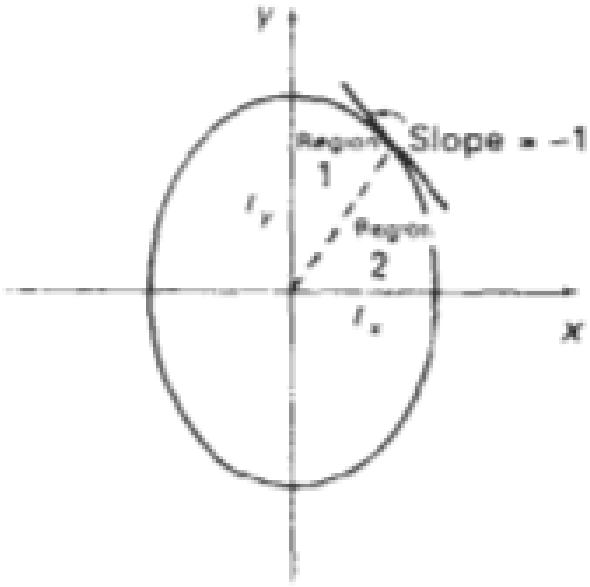


Figure 3-20
Ellipse processing regions. Over region 1, the magnitude of the ellipse slope is less than 1; over region 2, the magnitude of the slope is greater than 1.

At the boundary between region 1 and region 2, $dy/dx = -1$ and

$$2r_y^2 x = 2r_x^2 y$$

Therefore, we move out of region 1 whenever

$$2r_y^2 x \geq 2r_x^2 y$$

$$p1_k = f_{\text{ellipse}}\left(x_k + 1, y_k - \frac{1}{2}\right)$$

$$= r_y^2(x_k + 1)^2 + r_x^2\left(y_k - \frac{1}{2}\right)^2 - r_x^2 r_y^2$$

$$p1_{k+1} = f_{\text{ellipse}}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$

$$= r_y^2[(x_k + 1) + 1]^2 + r_x^2\left(y_{k+1} - \frac{1}{2}\right)^2 - r_x^2 r_y^2$$

$$p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2\left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right]$$

$$\text{increment} = \begin{cases} 2r_y^2 x_{k+1} + r_y^2, & \text{if } p1_k < 0 \\ 2r_y^2 x_{k+1} + r_y^2 - 2r_x^2 y_{k+1}, & \text{if } p1_k \geq 0 \end{cases}$$

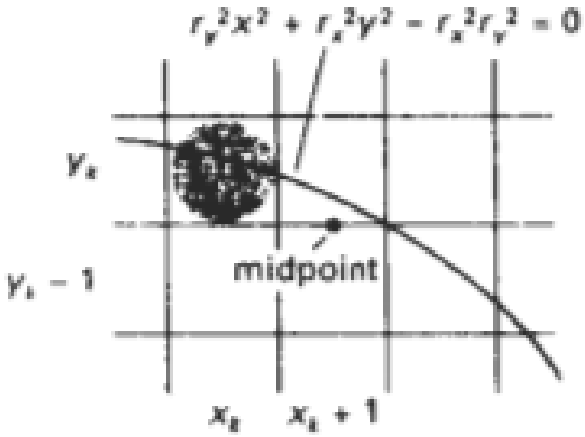


Figure 3-21
Midpoint between candidate pixels at sampling position $x_k + 1$ along an elliptical path.

$$2r_y^2 x = 0$$

$$2r_x^2 y = 2r_x^2 r_y$$

$$p1_0 = f_{ellipse}\left(1, r_y - \frac{1}{2}\right)$$

$$= r_y^2 + r_x^2\left(r_y - \frac{1}{2}\right)^2 - r_x^2 r_y^2$$

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

$$p2_k = f_{ellipse}\left(x_k + \frac{1}{2}, y_k - 1\right)$$

$$= r_y^2\left(x_k + \frac{1}{2}\right)^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2$$

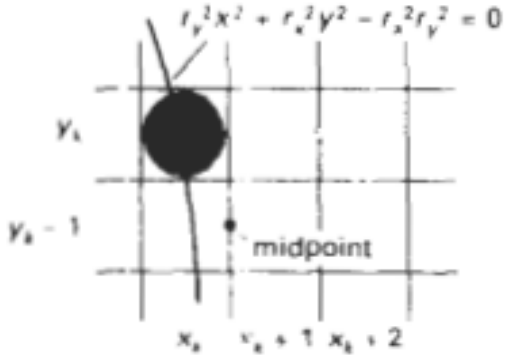


Figure 3-22
Midpoint between candidate pixels at sampling position $y_k - 1$ along an elliptical path.

$$p2_{k+1} = f_{\text{ellipse}}\left(x_{k+1} + \frac{1}{2}, y_{k+1} - 1\right)$$

$$= r_y^2\left(x_{k+1} + \frac{1}{2}\right)^2 + r_x^2[(y_k - 1) - 1]^2 - r_x^2r_y^2$$

$$p2_{k+1} = p2_k - 2r_x^2(y_k - 1) + r_x^2 + r_y^2\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right]$$

$$p2_0 = f_{ellipse}\left(x_0 + \frac{1}{2}, y_0 - 1\right)$$

$$= r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2$$

- To simplify the calculation of p20, we could select pixel positions in counterclock-wise order starting at (rx, 0).

- Unit steps would then be taken in the positive y direction up to the last position selected in region 1.

MidPoint Ellipse drawing algorithm…

Midpoint Ellipse Algorithm

1. Input r_x, r_y, and ellipse center (x_c, y_c), and obtain the first point on an ellipse centered on the origin as

$$(x_0, y_0) = (0, r_y)$$

2. Calculate the initial value of the decision parameter in region 1 as

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

3. At each x_k position in region 1, starting at $k = 0$, perform the following test: If $p1_k < 0$, the next point along the ellipse centered on $(0, 0)$ is (x_{k+1}, y_k) and

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2$$

Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ and

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

with

$$2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2, \qquad 2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2$$

and continue until $2r_y^2 x \geq 2r_x^2 y$.

4. Calculate the initial value of the decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as

$$p2_0 = r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

5. At each y_k position in region 2, starting at $k = 0$, perform the following test: If $p2_k > 0$, the next point along the ellipse centered on $(0, 0)$ is $(x_k, y_k - 1)$ and

$$p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2$$

Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ and

$$p2_{k+1} = p2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

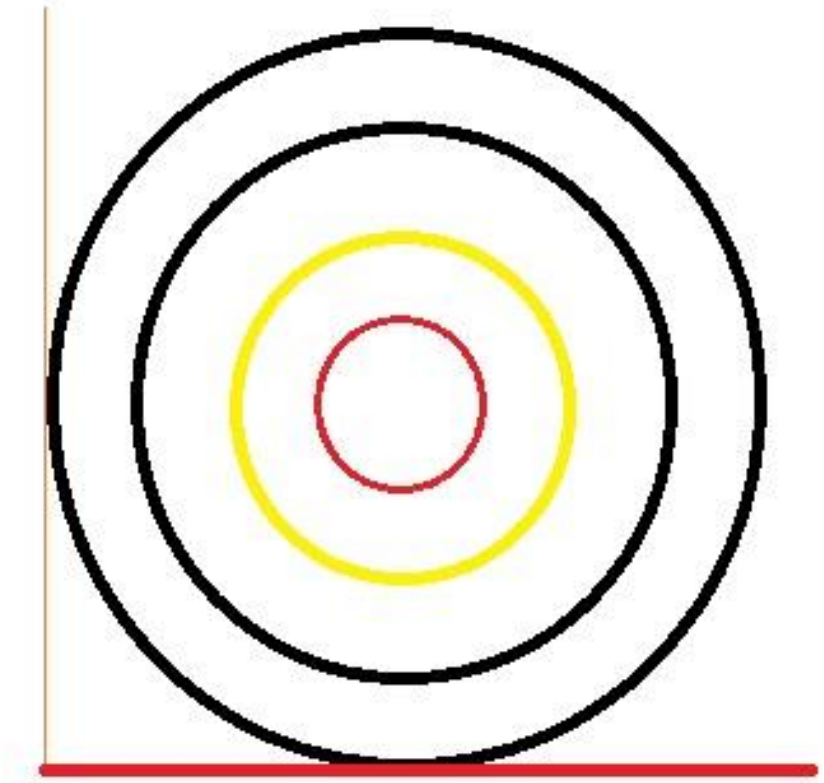
using the same incremental calculations for x and y as in region 1.

6. Determine symmetry points in the other three quadrants.
7. Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values:

$$x = x + x_c, \qquad y = y + y_c$$

8. Repeat the steps for region 1 until $2r_y^2 x \geq 2r_x^2 y$.

Lab4 Assignment for the Lab: write an OpenGL program for displaying the following diagram:



Scan-conversion of conic section

$$B^2 - 4AC \begin{cases} < 0, & \text{generates an ellipse (or circle)} \\ = 0, & \text{generates a parabola} \\ > 0, & \text{generates a hyperbola} \end{cases}$$

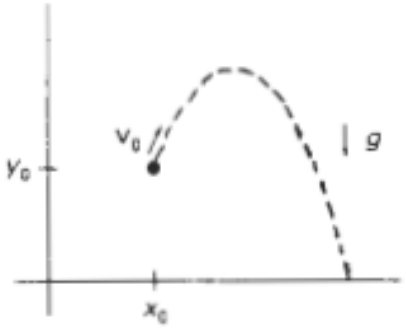


Figure 3-24
Parabolic path of an object tossed into a downward gravitational field at the initial position (x_0, y_0).

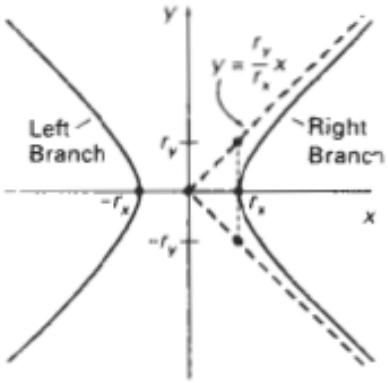


Figure 3-25
Left and right branches of a hyperbola in standard position with symmetry axis along the x axis.

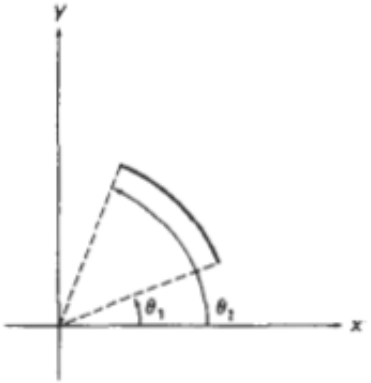


Figure 3-27
Circular arc specified by beginning and ending angles. Circle center is at the coordinate origin.

Area Filling

- Rasterizing a Polygon.
- A polyline is a chain of connected line segments. It is specified by giving the vertices or nodes P0,P1,P2,… and so on.
- The first vertex is called the initial or starting point and the last vertex is called the final or terminal point.
- When starting point and terminal point of any polyline is same, i.e. when polyline is closed then it is called polygon.

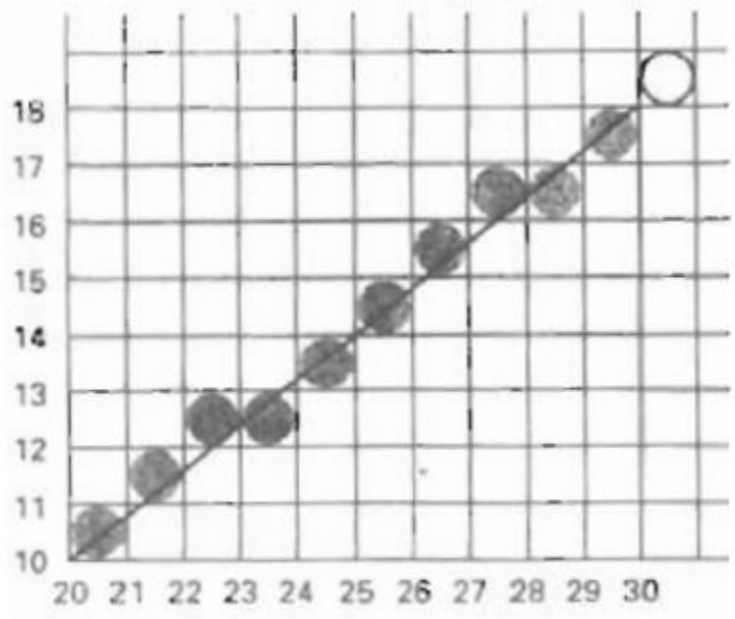


Figure 3-31
Line path and corresponding pixel display for input screen grid endpoint coordinates (20, 10) and (30, 18).

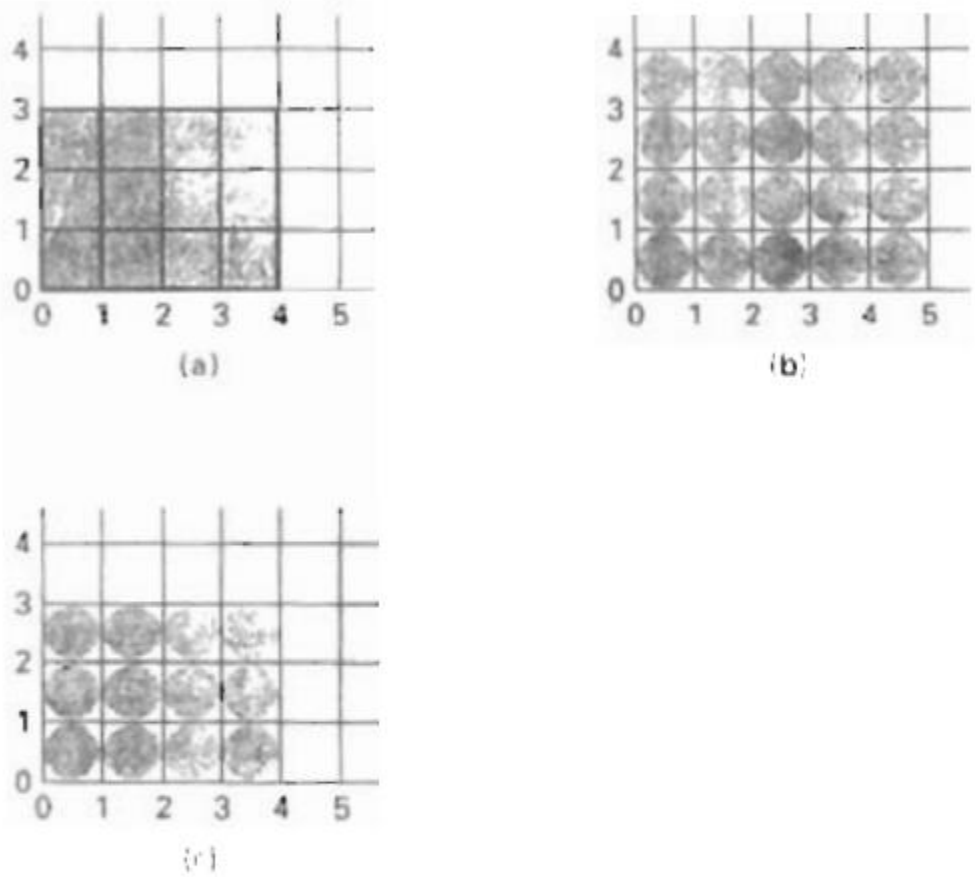
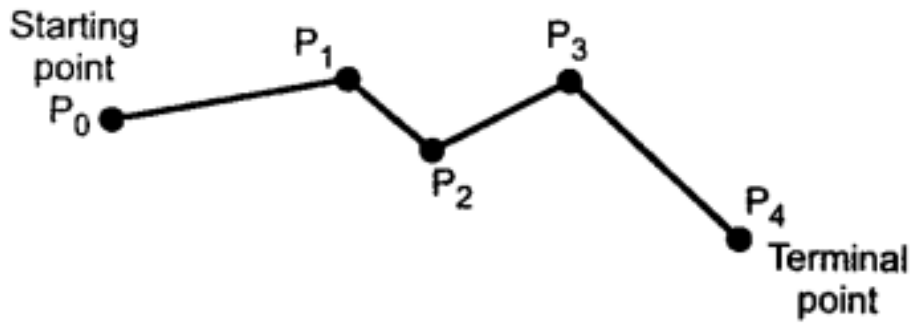
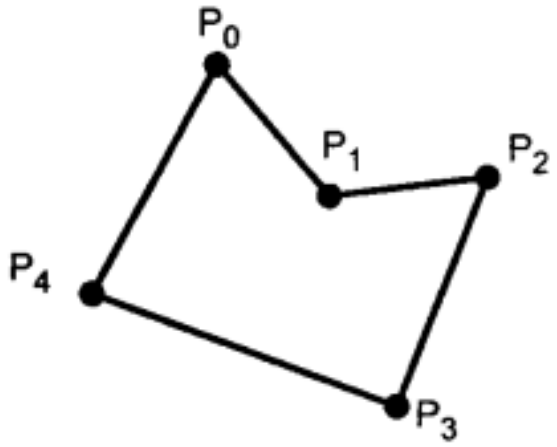


Figure 3-32
Conversion of rectangle (a) with verti-es at screen coordinates (0, 0), (4, 0), (4, 3), and (0, 3) into display (b) that includes the right and top boundaries and into display (c) that maintains geometric magnitudes.



(a) Polyline



(b) Polygon

Scan-converting a Polygon

- Generally closed contours are represented by a cluster of polygons.
- Thus, to fill or to draw a contour it is necessary to have methods for filling polygons.
- It may be observed that adjacent pixels are likely to have the same characteristics in polygon. This property is called spatial coherence.
- Adjacent pixels on a scan line are likely to have the same characteristics. This is called scan line coherence.

Types of Polygon

- **Convex polygon:** it is a polygon in which the line segment joining any two points within the polygon lies completely inside the polygon.

- **Concave polygon:** A concave polygon is a polygon in which the line segment joining any two points within the polygon may not lie completely inside the polygon.

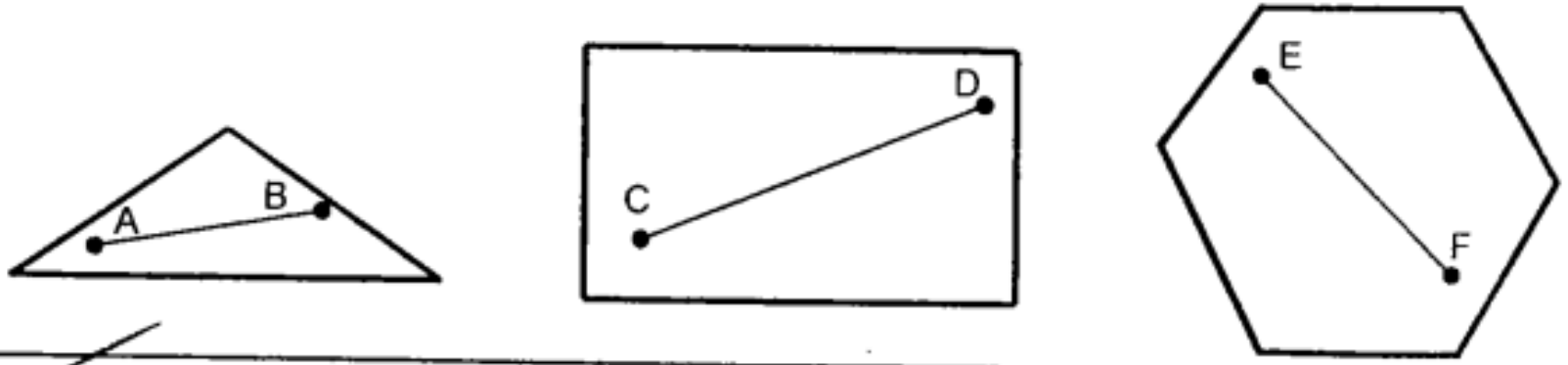


Fig. 3.2 Convex polygons

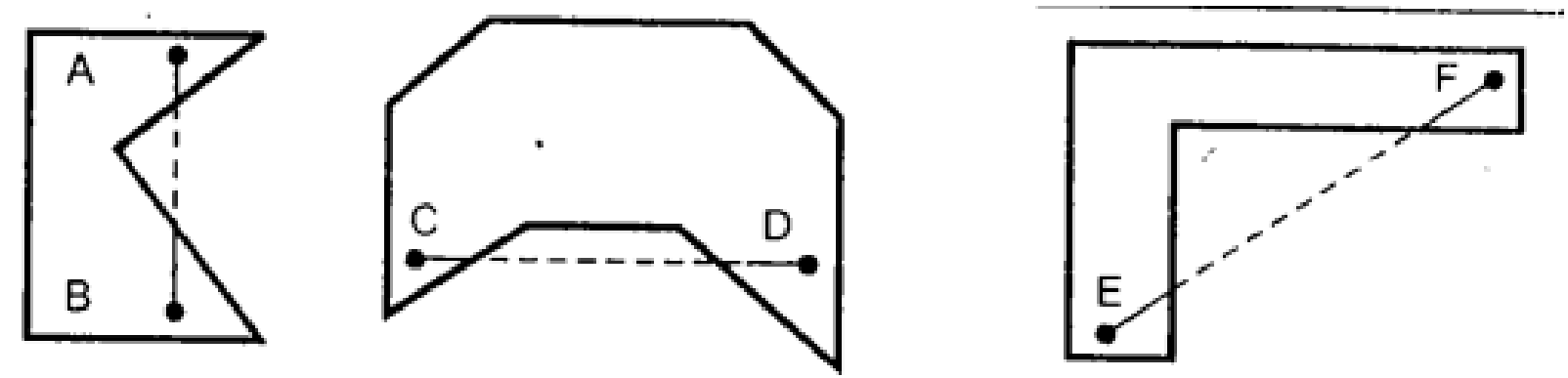
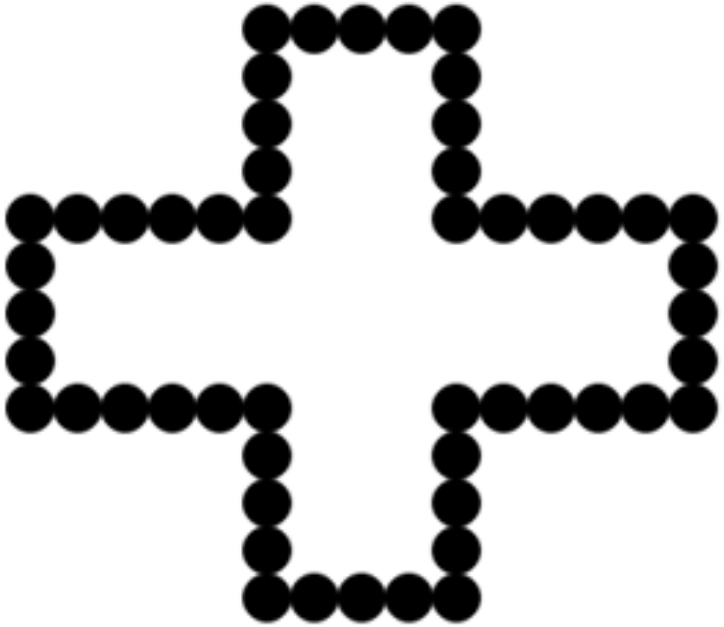


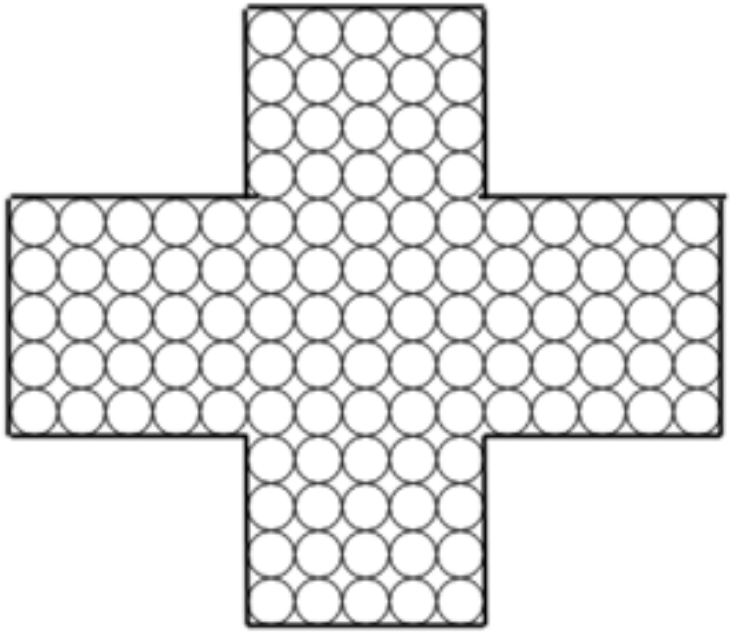
Fig. 3.3 Concave polygons

Filled-Area Primitives: Polygon filling

- Region filling is the process of filling image or region. Filling can be of boundary or interior region.
- Filling the polygon means highlighting all the pixels which lie inside the polygon with any color other than background color.
- Boundary Fill algorithms are used to fill the boundary and flood-fill algorithm are used to fill the interior.
- Seed fill methods



Boundary Filled Region



Interior or Flood Filled Region

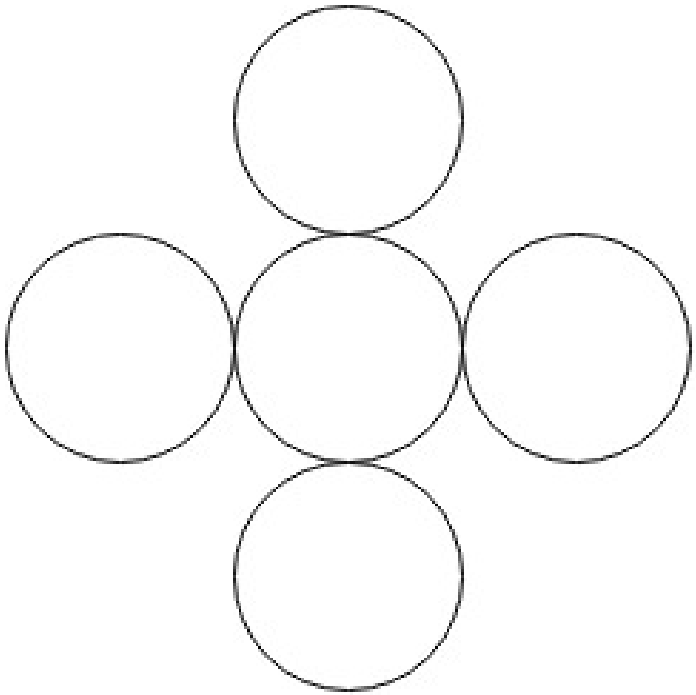
Seed Fill

- The seed fill algorithm is further classified as flood fill algorithm and boundary fill algorithm.
- An algorithms that fill interior-defined regions are called flood-fill algorithm.
- An algorithms that fill boundary-defined regions are called boundary fill algorithm or edge-fill algorithms.
  - Boundary fill
  - Flood fill

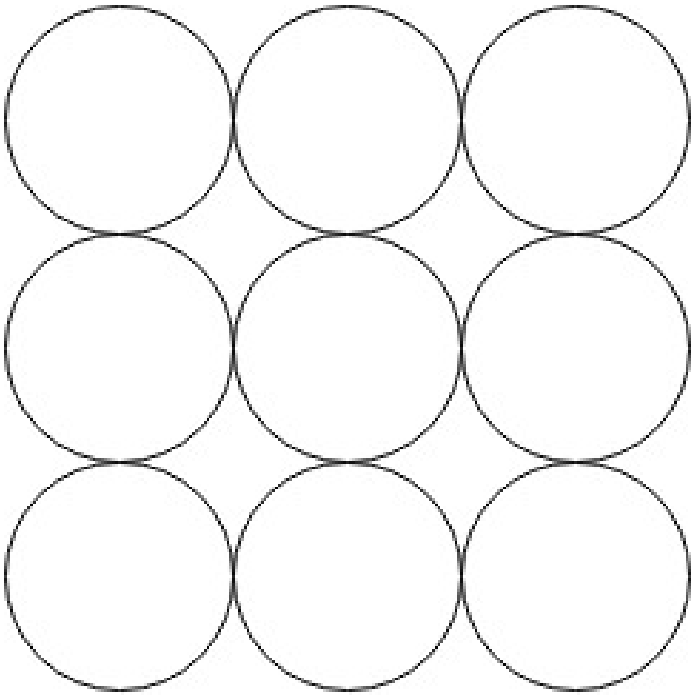
1. Boundary Filled Algorithm

- This algorithm uses the recursive method. First of all, a starting pixel called as the seed is considered.
- The algorithm checks boundary pixel or adjacent pixels are colored or not.
- If the adjacent pixel is already filled or colored then leave it, otherwise fill it.
- The filling is done using four connected or eight connected approaches.
- Four connected approaches is more suitable than the eight connected approaches.

Figure 3-42
Example color boundaries for a boundary-fill procedure.

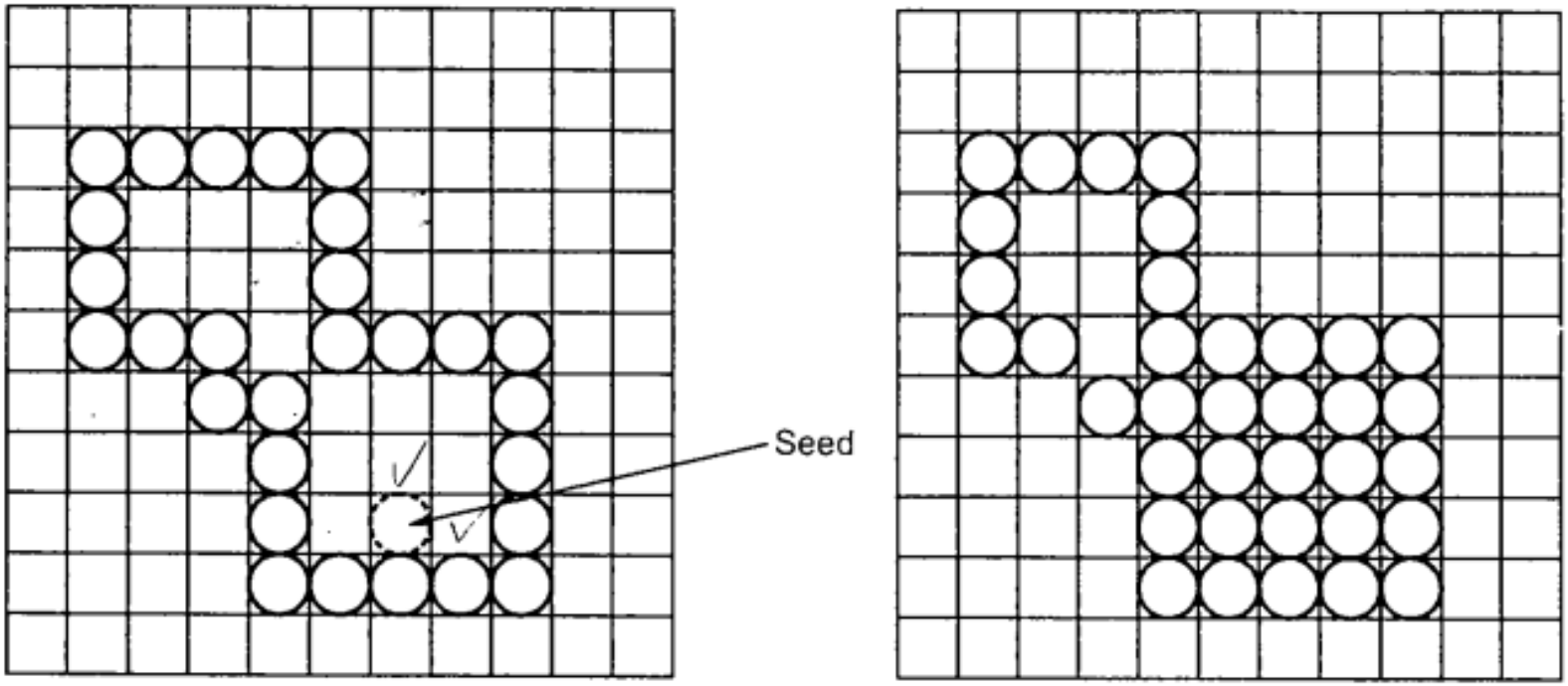


Four Connected



Eight Connected

- **Four connected approaches:** In this approach, left, right, above, below pixels are tested.
- **Eight connected approaches:** In this approach, left, right, above, below and four diagonals are selected.
- Boundary can be checked by seeing pixels from left and right first. Then pixels are checked by seeing pixels from top to bottom.
- The algorithm takes time and memory because some recursive calls are needed.


Seed

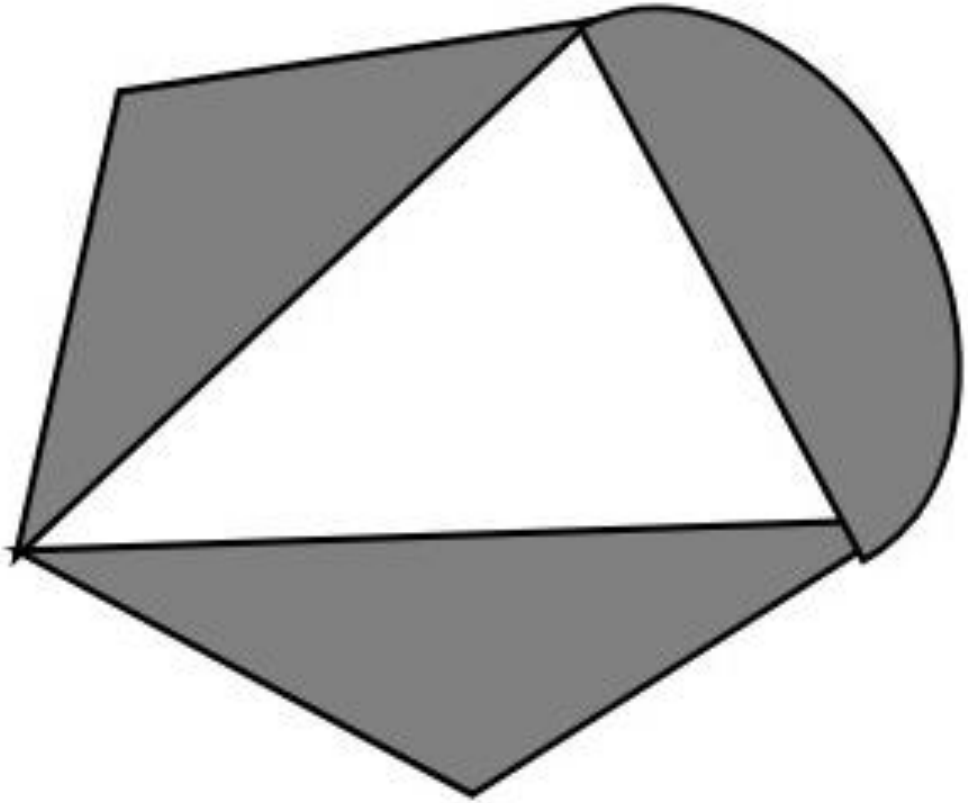
Boundary fill algorithm…

```
void boundaryFill4 (int x, int y, int fill, int boundary)
{
  int current;

  current = getPixel (x, y);
  if ((current != boundary) && (current != fill)) {
    setColor (fill);
    setPixel (x, y);
    boundaryFill4 (x+1, y, fill, boundary);
    boundaryFill4 (x-1, y, fill, boundary);
    boundaryFill4 (x, y+1, fill, boundary);
    boundaryFill4 (x, y-1, fill, boundary);
  }
}
```

2. Flood Fill Algorithm

- In this method, a point or seed which is inside region is selected. This point is called a seed point.
- Then four connected approaches or eight connected approaches is used to fill with specified color.
- The flood fill algorithm has many characters similar to boundary fill. But this method is more suitable for filling multiple colors boundary.
- When boundary is of many colors and interior is to be filled with one color we use this algorithm.



- In fill algorithm, we start from a specified interior point (x, y) and reassign all pixel values are currently set to a given interior color with the desired color.
- Using either a 4-connected or 8-connected approaches, we then step through pixel positions until all interior points have been repainted.

- Disadvantage:
  - Very slow algorithm
  - May be fail for large polygons
  - Initial pixel required more knowledge about surrounding pixels.

Flood fill algorithm…

```
void floodFill4 (int x, int y, int fillColor, int oldColor)
{
  if (getPixel (x, y) == oldColor) {
    setColor (fillColor);
    setPixel (x, y);
    floodFill4 (x+1, y, fillColor, oldColor);
    floodFill4 (x-1, y, fillColor, oldColor);
    floodFill4 (x, y+1, fillColor, oldColor);
    floodFill4 (x, y-1, fillColor, oldColor);
  }
}
```

Scan Line Polygon Fill Algorithm

- This algorithm lines interior points of a polygon on the scan line and these points are done on or off according to requirement.
- The polygon is filled with various colors by coloring various pixels.
- First of all, scanning is done. Scanning is done using raster scanning concept on display device.
- The beam starts scanning from the top left corner of the screen and goes toward the bottom right corner as the endpoint.

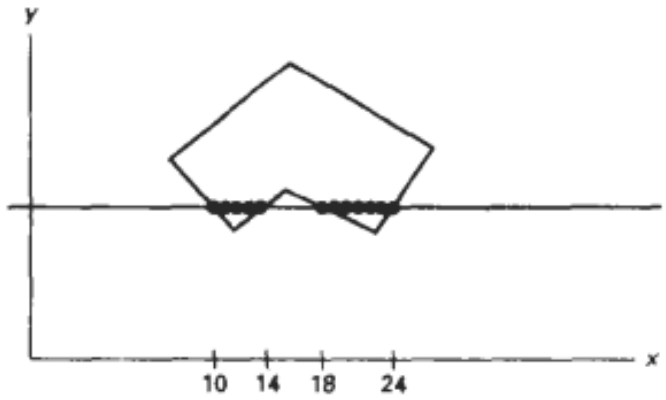


Figure 3-35
Interior pixels along a scan line passing through a polygon area.

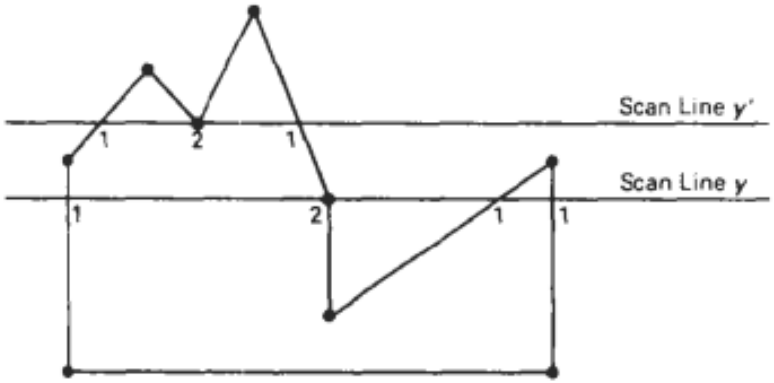
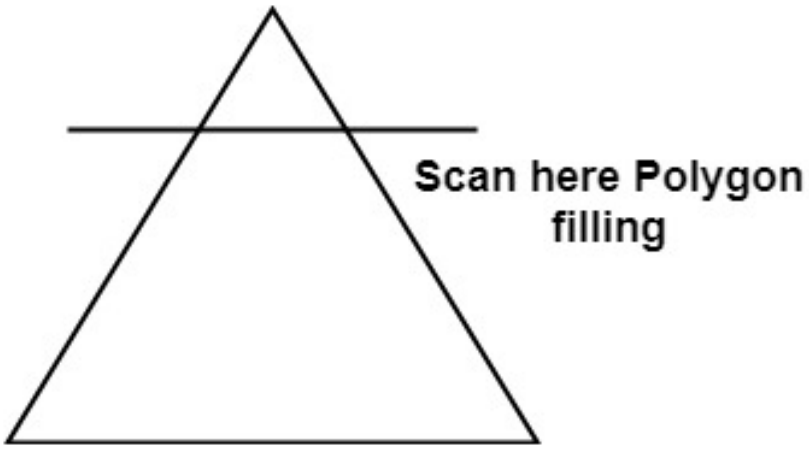
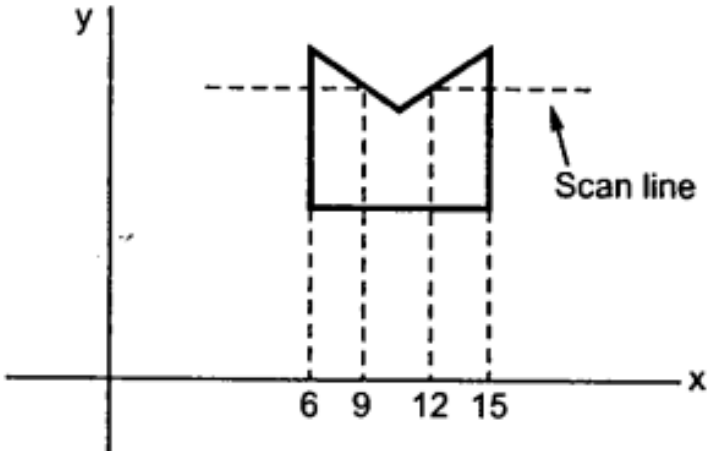


Figure 3-36
Intersection points along scan lines that intersect polygon vertices. Scan line y generates an odd number of intersections, but scan line y' generates an even number of intersections that can be paired to identify correctly the interior pixel spans.

- The algorithms find points of intersection of the line with polygon while moving from left to right and top to bottom.
- The various points of intersection are stored in the frame buffer.
- The intensities of such points is keep high.
- Concept of coherence property is used. According to this property if a pixel is inside the polygon, then its next pixel will be inside the polygon.





- Side effects of Scan Conversion:
  - **Staircase or Jagged:** Staircase like appearance is seen while the scan was converting line or circle.
  - **Unequal Intensity:** It deals with unequal appearance of the brightness of different lines. An inclined line appears less bright as compared to the horizontal and vertical line.

Scan line algorithm for polygon filling

1. Read n, the number of vertices of polygon
2. Read x and y coordinates of all vertices in array x[n] and y[n].
3. Find y_{min} and y_{max}.
4. Store the initial x value (x_1) y values y_1 and y_2 for two endpoints and x increment Δx from scan line to scan line for each edge in the array edges [n] [4].

   While doing this check that $y_1 > y_2$, if not interchange y_1 and y_2 and corresponding x_1 and x_2 so that for each edge, y_1 represents its maximum y coordinate and y_2 represents it minimum y coordinate.
5. Sort the rows of array, edges [n] [4] in descending order of y_1, descending order of y_2 and ascending order of x_2.
6. Set $y = y_{max}$
7. Find the active edges and update active edge list :

   if ($y > y_2$ and $y \leq y_1$)

   { edge is active }

   else

{ edge is not active }

8. Compute the x intersects for all active edges for current y value [initially x-intersect is x_1 and x intersects for successive y values can be given as

   $x_{i+1} \leftarrow x_i + \Delta x$

   where $\Delta x = -\frac{1}{m}$ and $m = \frac{y_2 - y_1}{x_2 - x_1}$ i.e. slope of a line segment

9. If x intersect is vertex i.e. x-intersect = x_1 and $y = y_1$ then apply vertex test to check whether to consider one intersect or two intersects. Store all x intersects in the x-intersect [ ] array.
10. Sort x-intersect [ ] array in the ascending order,
11. Extract pairs of intersects from the sorted x-intersect [ ] array.
12. Pass pairs of x values to line drawing routine to draw corresponding line segments
13. Set $y = y - 1$
14. Repeat steps 7 through 13 until $y \geq y_{min}$.
15. Stop